MÁSTER UNIVERSITARIO EN INGENIERÍA DE SISTEMAS Y DE CONTROL POR LA UNIVERSIDAD COMPLUTENSE DE MADRID Y LA UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

Trabajo Fin de Máster

TÍTULO: Modelado y simulación del micro vehículo aéreo Crazyflie 2.0

AUTOR	Pablo Antón Jornet
DIRECTOR	José Manuel Díaz Martínez
CURSO	2016-2017
CONVOCATORIA	Junio

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

MÁSTER UNIVERSITARIO EN INGENIERÍA DE SISTEMAS Y DE CONTROL POR LA UNIVERSIDAD COMPLUTENSE DE MADRID Y LA UNIVERSIDAD NACIONAL DE EDUCACIÓN A DISTANCIA

Modelado y simulación del micro vehículo aéreo Crazyflie 2.0

Proyecto tipo A: Proyecto específico propuesto por un profesor

AUTOR	Pablo Antón Jornet
DIRECTOR	José Manuel Díaz Martínez
CURSO	2016-2017
CONVOCATORIA	Junio

Espacio reservado para la hoja de calificaciones, que será introducida por la secretaría del Master después de la defensa del mismo

Autorización

Autorizamos a la Universidad Complutense y a la UNED a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firmado: Pablo Antón Jornet

Resumen del proyecto

Se han investigado ciertos aspectos del control semiautónomo de un vehículo aéreo no tripulado de reducidas dimensiones. Para ello se ha seguido la secuencia de tareas habitual en ingeniería de sistemas para los casos en los que se parte de un sistema ya existente. En primer lugar se ha establecido el modelo dinámico del sistema objeto de estudio. A continuación se ha simulado dicho modelo mediante ordenador, incluyendo el sistema de control desarrollado por el fabricante. Finalmente se ha verificado el funcionamiento correcto del modelo de simulación y se ha validado mediante la comparación con la dinámica del vehículo real.

Para llevar a cabo las tareas mencionadas se ha seleccionado como sistema objeto de estudio a la plataforma comercial Crazyflie 2.0. Se trata de un cuatrirotor de reducidas dimensiones y peso, especialmente diseñado para docencia e investigación.

El modelo matemático de la dinámica del micro vehículo aéreo se ha establecido a partir de principios físicos, mediante ecuaciones diferenciales. Para ello ha sido necesario realizar previamente un estudio de las características del vehículo.

El modelo de simulación por ordenador se ha realizado en Simulink®, plasmando las ecuaciones diferenciales del modelo matemático y reproduciendo el sistema de control implementado por el fabricante. Para ello ha sido necesario realizar previamente un estudio del firmware desarrollado por el fabricante del vehículo.

Se ha creado un simulador configurable en MATLAB® que permite introducir de forma cómoda la localización deseada del vehículo a lo largo del tiempo, bien de forma analítica o bien mediante captura de datos por telemetría, y que lanza, en base a dicha localización deseada, el modelo de simulación, produciendo posteriormente los resultados gráficos correspondientes.

El simulador se ha verificado y validado para asegurar que proporciona datos fiables.

El simulador permite, asimismo, el desarrollo de nuevos controladores para el vehículo y la realización de pruebas de los mismos mediante simulación, sin necesidad de modificar el firmware del controlador del vehículo para realizarlas, reduciendo el tiempo de desarrollo y evitando posibles accidentes del prototipo en el entorno real.

Lista de palabras clave

Cuatrirotor, modelado, simulador, Crazyflie 2.0, telemetría

Contenido

LISTA DE FIGURAS

LISTA DE TABLAS

LISTA DE SÍMBOLOS

(B)	Sistema de referencia ligado al vehículo B ≡ $(O_B X_B Y_B Z_B)$.
(E)	Sistema de referencia del laboratorio E ≡ $(O_E X_E Y_E Z_E)$,
a	Gradiente térmico en la tropopausa (atmosfera standard)
C_d	Coeficiente de resistencia del elemento de pala
C_l	Coeficiente de sustentación del elemento de pala
C_{m0}	Coeficiente de par lineal
C_{m1}	Coeficiente de par cuadrático
C_{t0}	Coeficiente de tracción lineal
C_{t1}	Coeficiente de tracción cuadrático
d	Distancia del eje del actuador al eje X_B (o al eje Y_B)
d'	Distancia del centro geométrico al eje del actuador
F_3	Componente de la fuerza aerodinámica en el eje Z_B
$\boldsymbol{F}_E$	Fuerza resultante aplicada, en el sistema de referencia (E)
F_x, F_y, F_z	Componentes de la fuerza resultante aplicada, en el sistema de referencia (E)
g	Aceleración de la gravedad
h	Altitud de vuelo
h_0	Altitud de referencia
I	Matriz de inercia, en el sistema de referencia (B)
I	Intensidad en el circuito del motor
I_{xx}, I_{yy}, I_{zz}	Momentos de inercia del vehículo, en el sistema de referencia (B)
K_m	Ganancia del motor
K_b	Constante del motor
L	Inducción del circuito del motor
m	Masa del vehículo
M	Par resistente de la hélice
M_1, M_2, M_3	Componentes del momento aerodinámico, en el sistema referencia (B)
$\boldsymbol{M}_B$	Momento resultante aplicado en el sistema de referencia (B)
M_x, M_y, M_z	Componentes del momento resultante aplicado, en sistema de referencia (B)
O_B	Origen del sistema de referencia ligado al vehículo (B)
O_E	Origen del sistema de referencia del laboratorio (E)
p	Componente x de la velocidad angular del vehículo, en sistema referencia (B)
p(h)	Presión a la altitud h
p_o	Presión de referencia
PWM_1	Voltaje con PWM enviado al motor 1
PWM_2	Voltaje con PWM enviado al motor 2
PWM_3	Voltaje con PWM enviado al motor 3
PWM_4	Voltaje con PWM enviado al motor 4
q	Componente y de la velocidad angular del vehículo, en sistema referencia (B)
R	Constante universal de los gases reales
R	Radio de la hélice

R	Resistencia del circuito del motor
r	Componente z de la velocidad angular del vehículo, en sistema referencia (B)
$\boldsymbol{R}_{EB}$	Matriz de rotación que transforma un vector del sistema (B) al sistema (E)
$\boldsymbol{R}_x(\phi)$	Matriz de rotación elemental (giro ángulo ϕ entorno al eje X)
$\boldsymbol{R}_y(\theta)$	Matriz de rotación elemental (giro ángulo θ entorno al eje Y)
$\boldsymbol{R}_z(\psi)$	Matriz de rotación elemental (giro ángulo ψ entorno al eje Z)
s	Variable compleja tras aplicar la transformada de Laplace
T	Tracción generada por la hélice
T_0	Temperatura de referencia
T_m	Constante de tiempo de los motores
U1	Fuerza aerodinámica deseada en el eje Z_B
U2	Momento aerodinámico deseado en el eje X_B
U3	Momento aerodinámico deseado en el eje Y_B
U4	Momento aerodinámico deseado en el eje Z_B
V_b	Fuerza contraelemotriz inducida en el circuito del motor
$\boldsymbol{v}_E$	Velocidad lineal del centro de masas del vehículo en el sistema (E)
$\mathrm{V}_1, \mathrm{V}_2, \mathrm{V}_3, \mathrm{V}_4$	Voltaje de mando motores 1, 2, 3 y 4
$\boldsymbol{W}_{EB}$	Matriz de transformación
$\mathbf{X}$	Vector de estado
x	Componente x de la posición del centro masas del vehículo en el sistema (E)
X_B	Eje X del sistema de referencia (B)
X_E	Eje X del sistema de referencia (E)
y	Componente y de la posición del centro masas del vehículo en el sistema (E)
Y_B	Eje Y del sistema de referencia (B)
Y_E	Eje Y del sistema de referencia (E)
z	Componente z de la posición del centro masas del vehículo en el sistema (E)
Z_B	Eje Z del sistema de referencia (B)
Z_E	Eje Z del sistema de referencia (E)
$\boldsymbol{\alpha}_B$	Aceleración angular del vehículo, en el sistema (B)
$\boldsymbol{\Gamma}_0$	Matriz de asignación lineal
$\boldsymbol{\Gamma}_1$	Matriz de asignación cuadrática
$\boldsymbol{\zeta}_E$	Posición del centro de masas del vehículo, en el sistema (E)
θ	Ángulo de balanceo
ρ	Densidad del aire
σ_e	Solidez efectiva de la hélice
$\boldsymbol{\Phi}$	Ángulos de Euler (Tait-Bryan)
ϕ	Ángulo de alabeo
ψ	Ángulo de guiñada
$\boldsymbol{\omega}_B$	Velocidad angular del vehículo, en el sistema (B)
$\omega_1, \omega_2, \omega_3, \omega_4$	Velocidad de rotación del eje de los motores 1,2,3,4

LISTA DE SIGLAS, ABREVIATURAS Y ACRÓNIMOS

API	Application Programming Interface
BET	Blade element theory
BIPM	International Bureau of Weights and Measures
BLE	Bluetooth low energy
CF1	Crazyflie
CF2	Crazyflie 2.0
DOF	Degrees Of Freedom
ECTS	European Credit Transfer and Accumulation System
EEPROM	Electrically Erasable Programmable Read-Only Memory
ESC	Electronic Speed Control
ETH	Eidgenössische Technische Hochschule
ETSII	Escuela Técnica Superior de Ingeniería Informática
EUR	Euro
FPV	First person view
GCC	GNU compiler collection
GPL	GNU General Public License
GPIO	General Purpose Input/Output
GPS	Global Positioning System
I2C	Inter-integrated circuit
IDE	Integrated Development Environment
IMU	Inertial Measuring Unit
ISM	Industrial, Scientific and Medical
JSON	JavaScript Object Notation
LSB	Less Significant Bit
MEMS	Microelectromechanical Systems
MOCAP	Motion Capture
OpenOCD	Open On-Chip-Debugger
PFM	Proyecto Fin de Master
PID	Proportional-Integral-Derivative
PWM	Pulse Width Modulation
RAM	Random Access Memory
RF	Radio Frequency
ROS	Robot operating system
SRAM	Static Random Access Memory
SLAM	Simultaneous Localization And Mapping
STM	Synchronous Transport Module
SWD	Serial Wire Debug
UART	Universal Asynchronous Receiver-Transmitter
UAV	Unmanned Aerial Vehicle
UI	User Interface
UNED	Universidad Nacional de Educación a Distancia
USB	Universal Serial Bus
USD	United States Dollar
UWB	Ultra Wide Band
VM	Virtual machine
ZMQ	Zero Em Queue asynchronous messaging library

1 INTRODUCCIÓN

En esta memoria se describen los detalles del trabajo de fin de master realizado dentro del Máster universitario en ingeniería de sistemas y de control de la Universidad Complutense de Madrid y la UNED. El trabajo constituye la asignatura final del Máster y tiene como objeto realizar una introducción a la metodología investigadora, y a la presentación de resultados de investigación. Tiene una carga lectiva de 12 créditos ECTS, es decir una dedicación aproximada de 300 a 360 horas, en la que se incluyen todos los aspectos de la investigación realizada, desde la compra y montaje del microdron, siguiendo por el estudio de sus especificaciones y del firmware del controlador implementado por el fabricante, pasando por su modelización y simulación, así como por la elaboración de los correspondiente programas principales y auxiliares, y finalizando por la verificación y validación del simulador. Asimismo, el tiempo dedicado a la elaboración de la presente memoria, que ha sido considerable, también está incluido en el cómputo de horas. En realidad se ha excedido ampliamente el número de horas programadas inicialmente para este trabajo debido a que la fase de depuración del simulador consumió una gran cantidad de tiempo.

2 OBJETIVOS

Se pretende desarrollar, verificar y validar un modelo de simulación del comportamiento dinámico del micro-vehículo aéreo Crazyflie 2.0 (CF2) [1]. Actualmente no se tiene conocimiento de la existencia de un modelo de simulación para dicho vehículo. El mencionado cuadrirotor está disponible de forma comercial, por lo que se ha comprado una unidad que se empleará en la verificación y validación del simulador. La principal ventaja de contar con un simulador del dron validado consiste en poder emplearlo en futuros desarrollos de sistemas de control del mismo sin necesidad de realizar ensayos con un prototipo físico, ya que el desarrollo se puede en ese caso considerar válido empleando únicamente el simulador. Esta forma de trabajar permite disminuir notablemente el tiempo y el coste de desarrollo de sistemas de control.

3 MOTIVACIÓN

El Crazyflie 2.0 está siendo empleado en la actualidad por numerosos grupos para investigar aspectos novedosos del control de los cuatrirotores. En la Tabla 1 se mencionan algunos de ellos.

Se trata de una plataforma de bajo peso y dimensiones que ha sido diseñada bajo un concepto abierto y flexible que cuenta con amplias posibilidades para su modificación y experimentación que la hacen ideal para su uso en investigación y desarrollo.

Tabla 1 : Grupos de investigación que usan el Crazyflie 2.0

Institución	**Área investigada**	**Ref.**
Lund Universtiy	-Estimación de posición 3D	[2]
ETH Zürich	-Implementación de algoritmos de estimación y control para una flota -Identificación de sistemas -Seguimiento de trayectorias mediante imagen -Fusión sensorial -Estimación de posición mediante señales de radio UWB	[3] [4] [5] [6] [7]
University of Twente	-Control robusto con evitación de colisiones usando el método de espacio de parámetros	[8]
Massachusetts Institute of Technology (MIT)	-Planificación y control de trayectorias para el vuelo en entornos densamente poblados de obstáculos usando control basado en modelo y técnicas IRIS y MISDP	[9]
University of California, Berkeley	-Aprendizaje automático de control mediante redes neuronales	[10]
École Polytechnique de Montréal	-Algoritmos de control de posición y trayectoria	[11]
McGill University	-Punteado con robots aéreos	[12]
University of Southern California	-Arquitectura de comunicaciones y estimación de estado para grandes enjambres volando en formación en interiores. -Realidad mixta	[13] [14]
Murray State University	-Vuelo autónomo	[15]
De La Salle University Manila	-Reconocimiento de imágenes mediante redes neuronales	[16]
Multi-Agent Autonomous Systems Lab	-Planificación de trayectorias en entornos densamente poblados de obstáculos	[17]

En la investigación de sistemas de control empleando el Crazyflie 2.0 se hace necesario, una vez se ha diseñado un nuevo controlador, modificar el firmware instalado en el microcontrolador del vehículo y grabarlo en la EEPROM, para posteriormente realizar las pruebas en vuelo pertinentes con objeto de comprobar las actuaciones del vehículo con el nuevo controlador. Este proceso iterativo de diseño de nuevos controladores se puede facilitar enormemente si se dispone de una herramienta de simulación que permita probar los nuevos diseños sin necesidad de realizar ensayos en vuelo. Una vez se ha diseñado un nuevo controlador, dicha herramienta puede permitir simularlo y ajustarlo en una fase previa, llegando a la fase de vuelos en prueba con un mayor conocimiento del funcionamiento del controlador, evitando así la mayoría de las pruebas en vuelo reales, que serían realizadas por simulación, y la posibilidad de dañar el vehículo en las mismas, en aquellos casos en los que el controlador no se ha diseñado correctamente.

Revisada la literatura, no parece haberse desarrollado una herramienta de dichas características para el caso concreto del Crazyflie 2.0 hasta ahora. Por tanto, se ha decidido desarrollar un prototipo sencillo de simulador para este vehículo en el presente trabajo. El desarrollo del

simulador constituye el núcleo del trabajo de fin de master, pero para poder completarlo se han tenido que realizar otra serie de tareas asociadas que también se describen en la presente memoria.

El proyecto elegido para el trabajo fin de master precisa para su ejecución del uso de la mayoría de las competencias adquiridas a lo largo del master en las distintas asignaturas del mismo. Al mismo tiempo permite explorar dos de los principales problemas que se pueden plantear en relación un sistema dinámico:

- El problema de la obtención del modelo del sistema: para el sistema objeto de estudio se puede obtener de forma sencilla un modelo matemático del mismo a partir de leyes físicas de la mecánica, la aerodinámica y la teoría de circuitos. Dicho modelo se debe validar experimentalmente una vez establecido. No es preciso para este caso llevar a cabo técnicas de identificación de sistemas ya que el modelado matemático genera un modelo lo suficientemente preciso para las tareas posteriores. La obtención de un modelo dinámico adecuado del sistema es crucial para los dos problemas siguientes: la simulación de su movimiento y el diseño y evaluación del control del mismo.
- El problema de la simulación del sistema: basándose en el modelo del sistema obtenido anteriormente se pretende crear una simulación por ordenador de la dinámica del sistema que pueda describir adecuadamente su comportamiento dinámico ante diferentes combinaciones de estados del sistema y entradas, proporcionándonos la evolución de la salida con el tiempo. El funcionamiento de la simulación por ordenador se debe validar experimentalmente.

4 SISTEMA A SIMULAR

4.1 Descripción

El Crazyflie 2.0 [1] es un micro vehículo aéreo basado en una placa de circuito impreso en la que se montan todos los componentes de los sistemas de alimentación y control del vehículo. A dicho circuito impreso se le añaden cuatro motores, con sus correspondientes soportes y hélices, y una batería, obteniéndose uno de los cuatrirotores de menor tamaño y peso de entre los disponibles comercialmente en la actualidad. De hecho, por sus reducidas dimensiones, el vehículo es definido por el fabricante como un nanocuatrirotor. En la Figura 1 se muestra el vehículo y se compara con el tamaño de una mano.

Figura 1 : Crazyflie 2.0 [1]

El vehículo se vende en forma de kit y está orientado a la docencia, investigación y desarrollo, estando disponible toda la información del mismo, incluidos los planos de las piezas y los esquemas de conexionado, en la página web del fabricante [18]. Asimismo, todo el software necesario para su funcionamiento ha sido colgado por la empresa diseñadora en GitHub [19] con una licencia de código abierto GPL [20].

La primera actividad que se ha llevado a cabo en el presente trabajo ha consistido es un estudio detallado de las características del vehículo, con objeto de estar en condiciones razonables de abordar su modelado, simulación, verificación y validación en fases posteriores. En los apartados que siguen se recogen los aspectos del estudio que se han considerado más relevantes para las tareas posteriores.

Asimismo otra actividad necesaria para alcanzar el objetivo planteado ha consistido en la adquisición y montaje del Crazyflie 2.0. y la instalación del software necesario para su funcionamiento tanto en la estación de tierra como en el mismo vehículo. Los aspectos más importantes de dichas actividades se han resumido, también en los apartados que siguen.

4.2 Especificaciones

En este capítulo se describen brevemente las principales características del microdron del que se va a desarrollar un modelo de simulación por ordenador. Se detallan tanto los aspectos mecánicos como electrónicos, cubriendo las especificaciones de los sensores y actuadores instalados en el vehículo, los componentes electrónicos que forman parte de los sistemas eléctrico, de control y de comunicaciones, así como las conexiones entre ellos. Asimismo se proporciona información de las distintas plataformas de expansión que se pueden añadir al vehículo para realizar depuraciones o añadir nuevas funcionalidades mediante la instalación de elementos adicionales. Ha sido necesario realizar este estudio previo de las características principales del sistema que se pretende simular, dado que se precisa tener un buen conocimiento de dicho sistema para poder realizar de forma adecuada el modelado matemático del mismo así como la posterior simulación y validación.

4.2.1 Dimensiones, masas y tiempos de operación

En la Tabla 2 se muestran las principales características geométricas del vehículo, así como las masas y los tiempos de operación

Tabla 2 : Características del Crazyflie 2.0

CARACTERÍSTICA	VALOR	
Dimensiones:		
-Generales	92x92x92 mm (de motor a motor)	
-Distancia entre ejes	39.73 mm (de motor a centro geométrico)	
Masas vehículo:		
-En vacío	27 g	[100 %]
-Máxima al despegue	42 g	
Masas componentes:		
-Motores y rotores	10.80 g = 2.70 g x 4	[40.0 %]
-Circuito impreso	7.22 g	[26.7 %]
-Batería	7.10 g	[26.3 %]
-Soportes motores	1.48 g = 0.37 g x 4	[5.5 %]
-Soporte batería	0.40 g	[1.5 %]
Máximo tiempo vuelo:		
-En vacío	7 min	
-A máxima carga	4 min	
Tiempo de carga batería:	40 min	
Alcance de la señal de control:		
-Crazyradio PA (2,4 GHz ; 100 mW)	1000 m (línea de visión)	
-Bluetooth Low Energy (2,4 GHz)	20 m	

Respecto a la masa del vehículo, se puede observar que para la operación normal del mismo, esta debe estar situada en el intervalo de 27 g a 42 g, correspondiendo el primer valor al vehículo completo pero sin ninguna placa de expansión o elemento adicional montado, mientras que el segundo valor se corresponde a la configuración para la cual la planta propulsora todavía puede proporcionar el suficiente empuje para contrarrestar el peso del vehículo y proporcionar la

aceleración necesaria para realizar un despegue. Por lo tanto, existe la posibilidad de instalar sensores adicionales y placas de expansión en el Crazyflie 2.0, pero con la limitación importante de que no pueden tener una masa superior a 15 g en su conjunto, ya que si se superara esa masa el despegue no sería posible. El incremento de masa del vehículo está asociado, asimismo, a la disminución del tiempo de vuelo, tal como se muestra en la Tabla 2. La diferencia entre la masa máxima al despegue y la masa en vacío (masa correspondiente a la comúnmente denominada carga de pago) proporciona un margen de flexibilidad para la realización de tareas de investigación y desarrollo con este microdron, permitiendo instalar y ensayar nuevos sensores y dispositivos en el mismo, o situar elementos reflectantes para la captura de movimiento.

En la Tabla 2 se muestran, asimismo, las contribuciones de los principales componentes a la masa en vacío del vehículo. Se puede observar el importante porcentaje que corresponde a los motores y la batería, que combinados constituyen el 66.3% de la masa en vacío, como suele ser el caso habitual de los microdrones. Este porcentaje es algo menor, de media, en cuadrirotores de mayores dimensiones.

4.2.2 Sensores y actuadores

En la Tabla 3 se presentan los sensores y actuadores con los que cuenta el vehículo.
Es interesante destacar que el dron cuenta con un número limitado de sensores que no le permite realizar de forma nativa vuelo autónomo. Esto es debido a que los diseñadores concibieron un vehículo de reducidas dimensiones y peso, teniendo que seleccionar para su instalación únicamente los sensores que eran estrictamente necesarios para su control por lo que su nivel de autonomía es reducido. Por ejemplo no cuenta con sensor GPS que le permitiría estimar la posición absoluta, ni con sensores de ultrasonidos, laser o infrarrojos para poder estimar las posiciones relativas, como la distancia al suelo o distancia a las paredes. No obstante, cuenta con una unidad inercial que integra giróscopo, acelerómetro y magnetómetro, y que le permite estimar la orientación del vehículo y las velocidades de giro asociadas a dicha orientación, así como las velocidades y aceleraciones lineales asociadas a la posición del vehículo. Asimismo tiene instalado un barómetro digital para la estimación de la altitud barométrica.

Tabla 3 : Sensores y actuadores del Crazyflie 2.0

SENSORES		ACTUADORES	
MPU-9250 [21]	-Giróscopo de 3 ejes -Acelerómetro de 3 ejes -Magnetómetro de 3 ejes	SKU 316030060 [22]	Motor CC sin núcleo (x4)
LPS25H [23]	Barómetro digital Termómetro	BC-CWP-01-A [24]	Rotor sentido horario (x2)
		BC-CCWP-01-A [24]	Rotor sentido antihorario (2x)

El campo de medida de los elementos de la unidad inercial debe ser seleccionado entre varios valores disponibles que son proporcionados por el fabricante del sensor. En la Tabla 4 se muestran los valores elegidos por la empresa diseñadora del Crazyflie 2.0.

Tabla 4 : Características de medida IMU (MPU-9250)

Elemento	Campo de medida	Sensibilidad (señal)	Sensibilidad (variable)
Giróscopo	±2000 º/s	16.4 LSB/(º/s)	61 m(º/s)/LSB
Acelerómetro	±8 g	4096 LSB/g	244 µg/LSB
Magnetómetro	±4800 µT	1.6 LSB/µT	0.6 µT/LSB

En la Figura 2 se muestran los sistemas de referencia y criterios de signos que emplea la unidad inercial para proporcionar sus datos. La unidad va instalada en la tarjeta impresa en una posición y orientación determinadas que están marcada gráficamente en dicha tarjeta tal y como podemos observar en la Figura 6 del apartado 4.2.4. No obstante, la empresa diseñadora del vehículo realizó cambios a dichos sistemas de referencia y criterios de signos a través del firmware del controlador, con el objeto de adecuarlos al sistema de referencia y criterio de signos habitual en aeronáutica. Desafortunadamente, el sistema de referencia y el criterio de signos finalmente empleados no son los habituales en aeronáutica debido a que el equipo diseñador tomó como criterio de signos el de la señal de referencia introducida a través de la estación de tierra, cuyo criterio es opuesto en el caso de cabeceo y guiñada, y olvidó invertir el eje vertical. Se entró en contacto con la empresa diseñadora a raíz de la aparente incongruencia entre lo marcado en la tarjeta impresa y lo empleado en el sistema de control, y la empresa confirmó los aspectos arriba indicados. La consecuencia de lo anterior es que en el resto del trabajo no se han adoptado los sistemas de referencia y criterios de signos mostrados en la Figura 2 y en la tarjeta impresa, sino los empleados tanto por el firmware del controlador como por el cliente de la estación de tierra. De esta manera se ha asegurado la congruencia de las señales y sus signos en todo el proyecto. En el apartado 6.3 se proporcionan más detalles del sistema de referencia y el criterio de signos empleado en el presente trabajo.

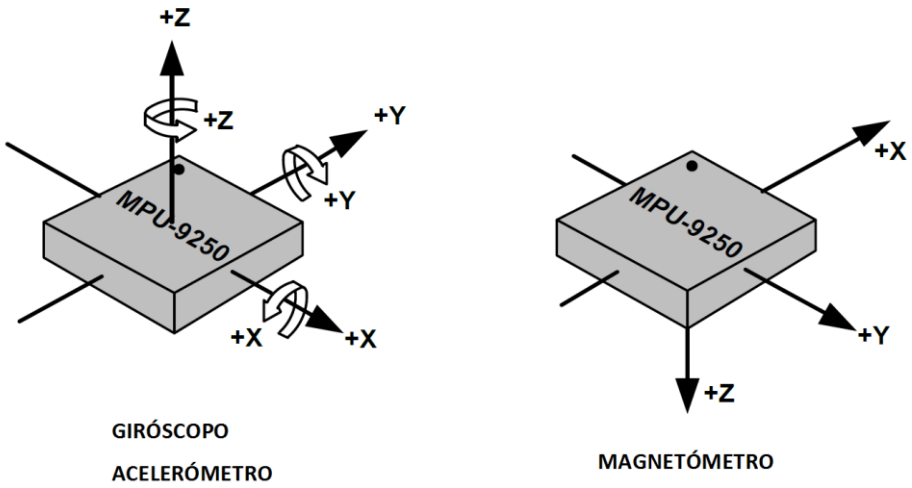


Figura 2 : Sistema de referencia de la unidad inercial MPU-9250 [21]

En lo que respecta al campo de medida del barómetro digital, los valores extraídos de su hoja de datos se muestran en la Tabla 5.

Tabla 5 : Características de medida barómetro digital (LPS25H)

Elemento	Campo de medida	Sensibilidad (señal)	Exactitud
Barómetro	260-1260 hPa	4096 LSB/hPa	±0.1 hPa (*)
Termómetro	0-80 ºC	480 LSB/ºC	±2 ºC (**)

(*) P=800-1100 hPa ; T = 25 ºC
(**) T=0-65 ºC

La exactitud del barómetro, que se emplea en la estimación de la altitud del vehículo, proporcionada por el fabricante es de ±10 Pa. Dicha exactitud la podemos evaluar en términos de altitud-presión usando las definiciones de la atmósfera standard internacional [25], tal y como se muestra en la ecuación (1).

$$p(h) = p_o \left(1 - \frac{a(h - h_0)}{T_0}\right)^{-\frac{g}{aR}} \quad (1)$$

Tomando la temperatura al nivel del mar y el gradiente de temperatura correspondiente a la troposfera, se puede obtener el valor de la exactitud del barómetro para la altitud-presión, empleando la ecuación (2).

$$h - h_0 = \frac{T_0}{a}\left(1 - e^{\frac{aRL\frac{p(h)}{p_0}}{g}}\right) = 0.83\ m \quad (2)$$

Es decir, que una vez establecida la equivalencia entre el valor de presión proporcionado por el sensor y el de altitud del vehículo en las condiciones habituales de uso, podemos observar que la exactitud en la estimación de altitud es de ±83 cm. Esta exactitud puede ser aceptable para vuelos en el exterior, pero es claramente insuficiente para ser usada por un controlador para mantenerla constante en vuelos en interiores, y especialmente para maniobras de precisión como pueden ser las operaciones de despegue y aterrizaje.

Para confirmar los cálculos anteriores se realizó una prueba estática, una vez ensamblado el Crazyflie 2.0, situándolo en una superficie fija y realizando telemetría de la altitud presión capturada por el microcontrolador. Los resultados del experimento se muestran en la Figura 3, donde puede apreciarse que los valores medidos fluctúan dentro de una banda de unos 80 cm, tal y como se esperaba.

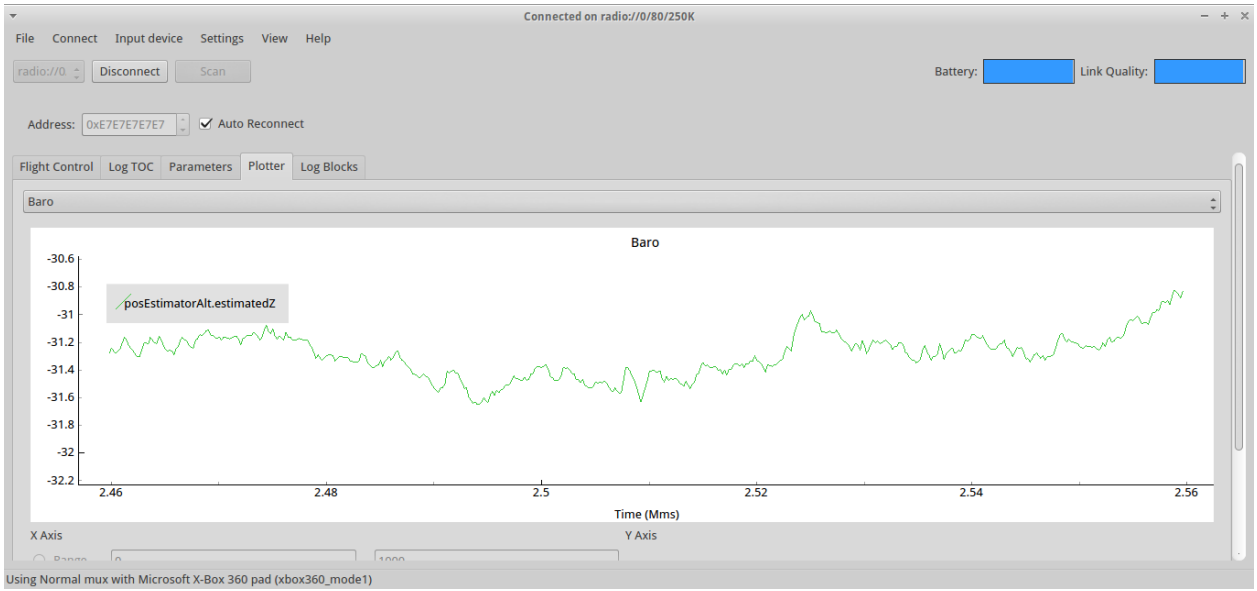


Figura 3 : Fluctuación de la altitud presión con vehículo estático

El vehículo está equipado con cuatro actuadores. Se trata de motores de corriente continua con escobillas y sin núcleo, cuyos elementos son parecidos a los mostrados en la Figura 4.

La principal característica de este tipo de motores es que contienen un imán permanente situado en el interior del elemento móvil, y que permanece fijo. Asimismo la parte móvil o rotor rodea a dicho imán permanente girando alrededor de él. El rotor no presenta devanados y se caracteriza por tener una masa muy reducida, lo que le proporciona una inercia muy baja y la posibilidad de alcanzar aceleraciones muy altas en breves espacios de tiempo. Es decir este tipo de motores presentan unas características dinámicas muy rápidas con tiempos de respuesta al controlador muy cortos, ideales para su uso en control de vuelo. Asimismo, al ser motores con escobillas, a diferencia de los motores sin escobillas que abundan en otros diseños de UAVs, el sistema de control es más sencillo ya que no se necesita un sistema electrónico de control de velocidad (ESC).

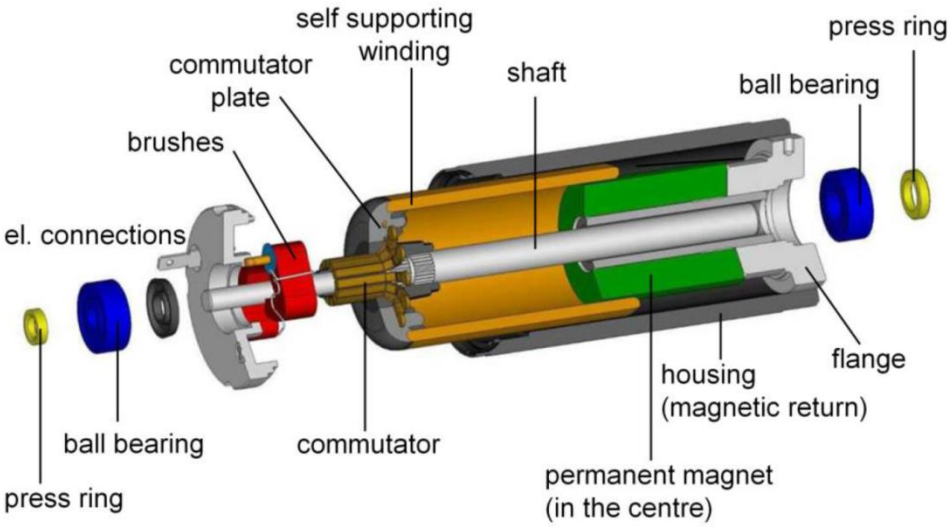


Figura 4 : Componentes principales del motor SKU 316030060 [26]

Finalmente las hélices son hélices de plástico standard que presentan cierta flexibilidad, permitiendo el bataneo.

4.2.3 Microcontroladores

El Crazyflie 2.0 emplea dos microcontroladores para gestionar los distintos sistemas del vehículo: el sistema eléctrico, el sistema de comunicaciones y el sistema de control de los actuadores, así como el sistema de adquisición de datos. La división de tareas se lleva a cabo de la siguiente forma:

- STM32F405 [27]: microcontrolador encargado de la adquisición de datos de los sensores de a bordo, estimación del estado del vehículo, generación de señales de control y control de los actuadores. Se trata de un ARM® Cortex™-M4 a 168MHz con 192kb de SRAM y 1Mb de memoria flash. Fabricante: STMicroelectronics.
- nRF51822 [28]: microcontrolador encargado de gestionar la energía eléctrica (control de la batería) y las comunicaciones con la estación de tierra a través de la antena. Se trata de un ARM® Cortex™-M0 a 32Mhz con 16kb de SRAM y 128kb de memoria flash. Fabricante: Nordic Semiconductor.

En la Figura 5 se muestra un esquema de la arquitectura del sistema.

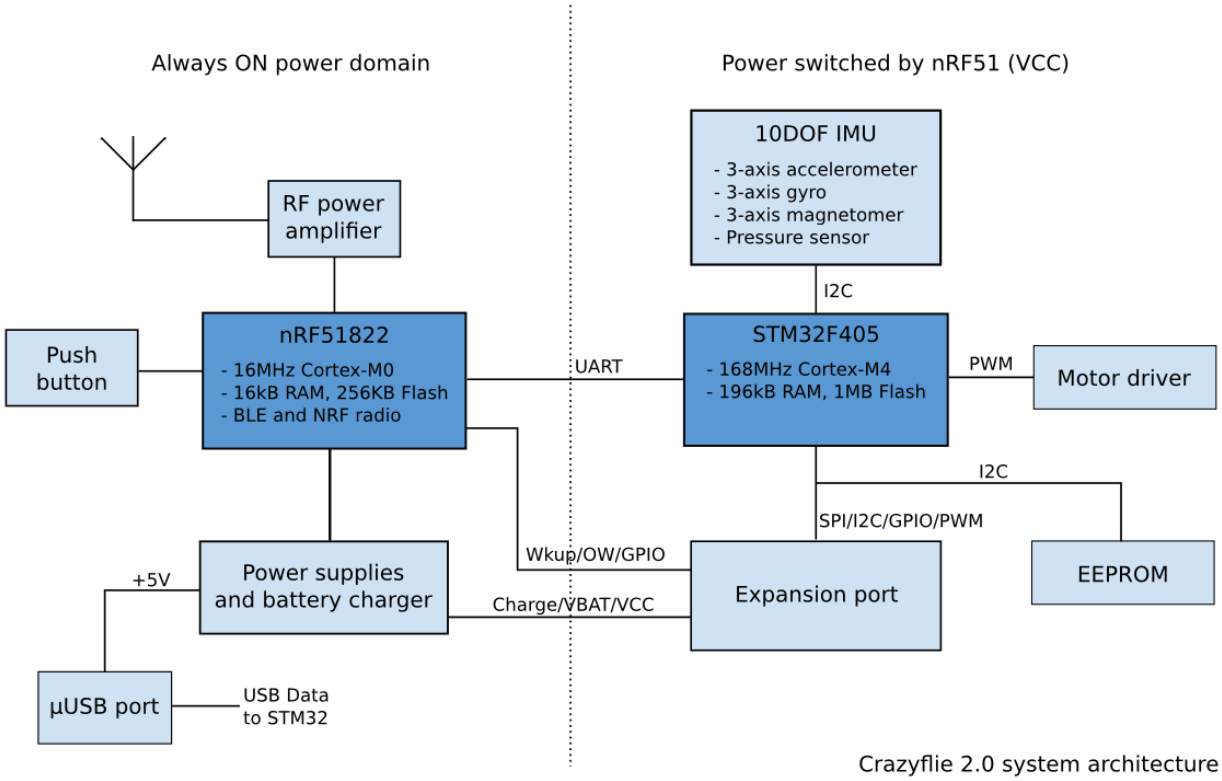


Figura 5 : Arquitectura del sistema de control y comunicación Crazyflie 2.0 [29]

Un aspecto fundamental del presente trabajo ha consistido en entender la forma en la que el microcontrolador STM32F405 fue integrado en la arquitectura del sistema y programado por la compañía fabricante. Esto ha sido así porque uno de los objetivos del trabajo consiste en identificar y simular, entre otros aspectos, el controlador de localización y el controlador de los

motores del vehículo, y ambos controladores están programados en el STM32F405. Así, el STM32F405 recibe las señales procedentes de los distintos sensores directamente a través de conexiones I2C y la señal de referencia a través de la conexión UART con el otro controlador de a bordo (que a su vez ha recibido dicha señal de referencia de la estación de tierra por medio de la antena de a bordo). El microcontrolador realiza la estimación del estado del vehículo fusionando las señales de los sensores. A partir del conocimiento del estado del vehículo y de la señal de referencia, el microcontrolador STM32F405 genera la señal de control en forma de voltaje con modulación de ancho de pulso (PWM) para cada uno de los cuatro actuadores.

Todas estas tareas que se han comentado anteriormente están codificadas en el código del firmware del microcontrolador desarrollado por el fabricante que está disponible, al igual que el resto del software del proyecto, en un repositorio de acceso libre [30]. Dicho software engloba conceptualmente al estimador de localización, al controlador de localización y al controlador de la planta de potencia, elementos que son fundamentales para la correcta operación del vehículo. Es por ello, que entender el funcionamiento del firmware del microcontrolador STM32F405 ha sido una tarea clave para poder alcanzar el objetivo del presente trabajo.

4.2.4 Localización de componentes

En la Figura 6 se muestra la vista superior del circuito impreso que constituye la principal estructura de soporte del Crazyflie 2.0. En la Figura 7 se muestra la vista inferior.

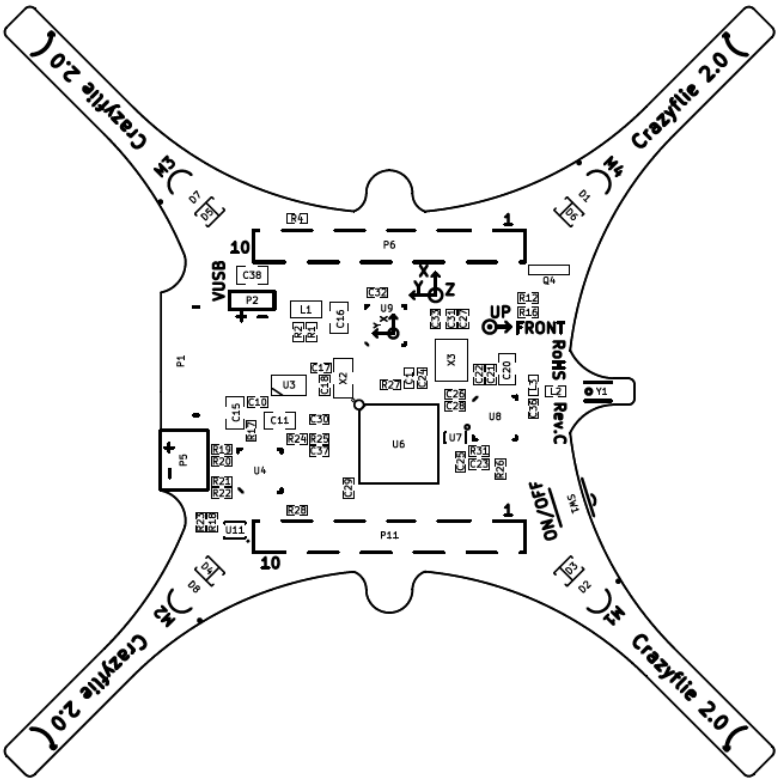


Figura 6 : Circuito impreso (vista superior) [31]

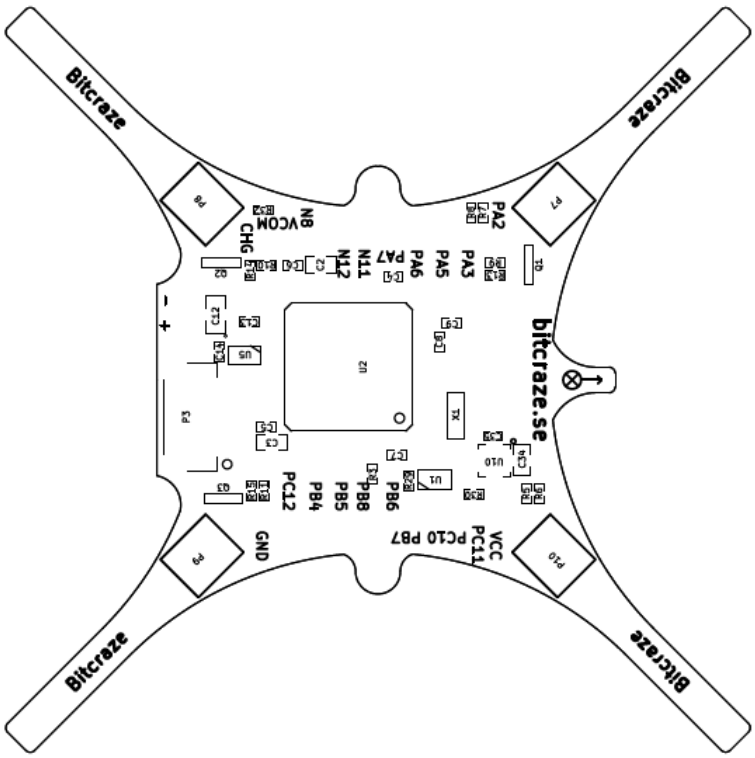


Figura 7 : Circuito impreso (vista inferior) [31]

En ambas figuras se puede observar la localización asignada a los distintos componentes que se montan en el circuito impreso. Asimismo, la Figura 6 muestra el sistema de referencia empleado por la unidad inercial que, tal y como se ha explicado en el apartado 4.2.2, no es el que se ha empleado en este trabajo. En la Tabla 6 se listan los principales componentes que van instalados en el circuito impreso, excluyendo a los componentes elementales (resistencias, condensadores, diodos, etc.). Los esquemas de conexionado de los componentes, así como los valores de los componentes elementales se encuentran disponibles en la página web del fabricante del dron [31].

Tabla 6 : Componentes principales del Crazyflie 2.0

Id	Descripción	Función
P1	MICRO-B-USB	Conector USB micro-B
P2	No instalado	Conector de alimentación externa
P3	CONN_STM_SWD	Conector STM (depuración)
P4	No instalado	conector nRF (depuración)
P5	CONN_BAT	Conector batería (Molex 51005-2P)
P6	CONN_10_LEFT	Puerto de expansión izquierdo
P7	CONN_2	Conector motor 1
P8	CONN_2	Conector motor 2
P9	CONN_2	Conector motor 3
P10	CONN_2	Conector motor 4
P11	CONN_10_RIGHT	Puerto de expansión derecho
U1	24AA64FT-E/OT	EEPROM de 8 KB
U2	STM32F405RG	Controlador principal MCU (Cortex-M4, 168MHz, 192kb, SRAM, 1Mb flash)
U3	NCP702SN30	Regulador de voltaje
U4	BQ24075	Cargador de batería
U5	LP2985-30	Regulador
U6	NRF51822	Controlador de comunicaciones y potencia eléctrica MCU (Cortex-M0, 32Mhz, 16kb SRAM, 128kb flash)
U7	BALUN	Balun de radiofrecuencia
U8	RFX2401C	Transceptor Zigbee 2.4 Ghz
U9	MPU9250	Acelerómetro, giróscopo y magnetómetro de 3 ejes
U10	LPS25H	Sensor de presión y temperatura
U11	SiP32431	Interruptor de carga con bloqueo inverso
X1		Cristal de cuarzo de 8Mhz
X2	No instalado	Cristal RTC opcional
X3		Cristal de cuarzo de 16Mhz
Y1		Antena

4.2.5 Placas de expansión

El vehículo admite el montaje de numerosas placas de expansión que se conectan a través de los puertos P6 y P11, a ambos lados de la placa base, con objeto de introducir nuevo hardware de una forma muy flexible. En el presente trabajo no se ha empleado ninguna placa de expansión centrándose la investigación en la configuración básica del vehículo.

4.2.6 Estación de tierra

La estación de tierra tiene la función principal de comunicar la señal de referencia a lo largo del tiempo al vehículo, es decir la posición y orientación deseadas. Asimismo, desde ella se realiza la telemetría de las variables de vuelo y también cuenta con la posibilidad de modificar los parámetros del controlador durante el vuelo. Finalmente, la estación se emplea también para cargar el firmware de los controladores del vehículo en las memorias correspondientes.

La estación de tierra se basa en un cliente software que se comunica con la antena del vehículo. La empresa diseñadora proporciona distintas posibilidades tanto para el cliente (basado en PC o basado en dispositivo móvil) como para la conexión de radiofrecuencia (radio banda ISM o Bluetooth) o para el mando de señal de referencia (dispositivo standard de juego o teclado). En la Tabla 7 se detalla la configuración que se ha empleado en el presente trabajo.

Tabla 7 : Elementos principales estación de tierra

Elemento	Características
Cliente	-Cliente para PC (Cfclient 2016.4-5-g8980a2a-modified) [32] -Instalado en ordenador portátil en máquina virtual Bitcraze VM 2016-06 [33]
Antena	-Radio Crazyradio PA USB 2.4 GHz banda ISM [34] -Insertada en puerto USB del portátil
Mando	-Controlador Xbox360 para Windows -Insertado en puerto USB del portátil

En la Figura 8 se muestra la estación de tierra empleada en el presente trabajo donde se pueden apreciar la antena, el mando y el cliente.

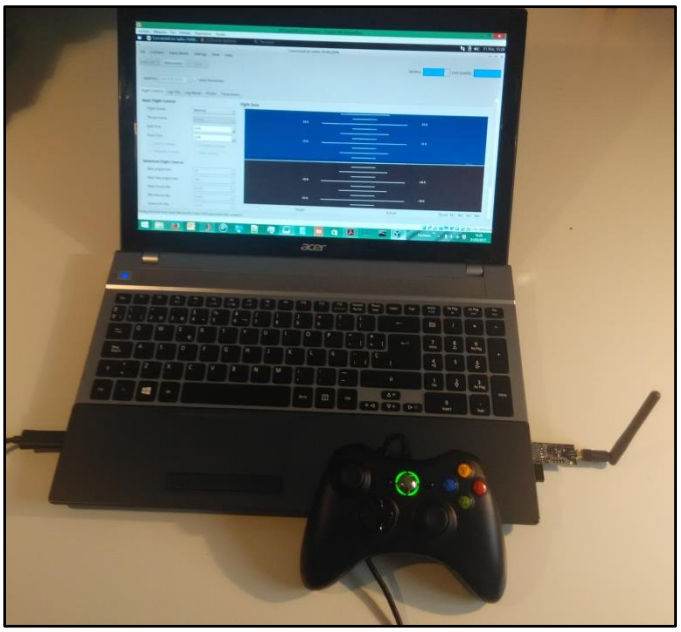


Figura 8 : Estación de tierra

4.2.7 Lenguajes de programación

El entorno empleado en la operación del vehículo se compone de diversos elementos software desarrollados o empleados por los diseñadores del mismo para facilitar su uso. En la Tabla 8 se resumen las principales aplicaciones necesarias y el lenguaje en las que están programadas.

Tabla 8 : Aplicaciones y lenguajes de programación

Aplicación	Lenguaje
Firmware microcontroladores vehículo	C
Firmware antena estación tierra	C
Cliente estación tierra (Win/Linux/OSX)	Python
Cliente dispositivos móviles Android	Java
Cliente dispositivos móviles iOS	Swift

A lo largo del presente trabajo se ha tenido que emplear distintas herramientas para llevar a cabo el amplio rango de actividades necesarias. En la Tabla 9 se listan las principales herramientas empleadas y se detalla brevemente el uso que se les ha dado.

Tabla 9 : Herramientas empleadas

Herramienta	Uso
IDE MATLAB® [35]+ Simulink® [36]	Simulador, procesado datos vuelo, verificación y validación
IDE eclipse© [37] + gcc [38]	Firmware STM32F405
GitHub [19]	Clonación repositorios

4.3 Adquisición y montaje

Un aspecto importante del presente trabajo ha consistido en la verificación y validación de la simulación por ordenador de la dinámica del vehículo, implementada a través del simulador desarrollado. Dicha actividad se describe en detalle en el capítulo 9. Para poderla llevar a cabo, se ha contado con un ejemplar real del vehículo que ha permitido poder comparar los datos obtenidos en pruebas reales con los obtenidos mediante simulación. En este capítulo se resumen las actividades previas que se realizaron para contar con un ejemplar de serie del Crazyflie 2.0.

4.3.1 Pedido

En enero de 2016 se realizó el pedido de un kit que contenía un ejemplar de Crazyflie 2.0 por piezas para ser montado, junto con la Crazyradio PA y alguna placa de expansión. El precio fue de 215 USD. El pedido fue distribuido por seeed studio [39], un fabricante chino de componentes electrónicos que es el principal distribuidor designado por la empresa diseñadora del producto

4.3.2 Recepción

Tras recibir el kit en un pequeño paquete procedente de China, se procedió a realizar la apertura del envío y la recepción de las piezas en mayo de 2016. El proceso se grabó y se subió a internet [40], siendo el vídeo, asimismo, parte de un proyecto de un estudiante de bachillerato, encaminado a crear un canal de videos tecnológicos en internet.

4.3.3 Montaje

El montaje del Crazylie 2.0 se realizó en enero de 2017, siguiendo las instrucciones de la web del fabricante [41]. El proceso fue muy sencillo sin presentar ninguna dificultad.

La Figura 9 muestra la placa de circuito impreso recibida y en la Figura 10 se ve como la placa pasa satisfactoriamente el test inicial al que hay que someterla antes del montaje.

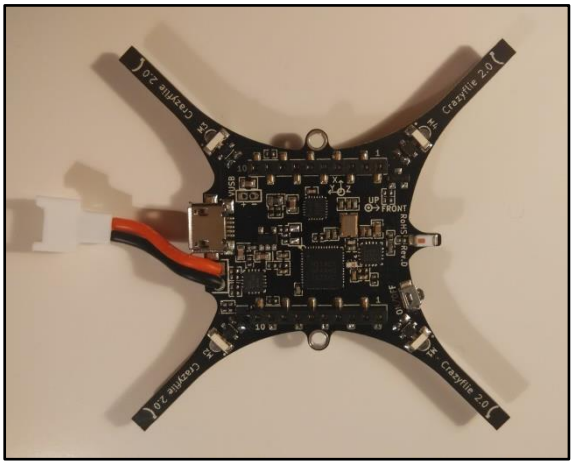


Figura 9 : Montaje CF2 (placa)

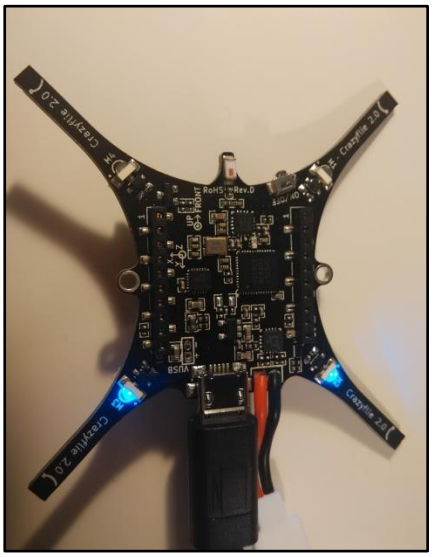


Figura 10 : Montaje CF2 (test placa)

En la Figura 11 se muestran los motores del kit una vez trenzados los cables de alimentación (el kit proporciona un motor de repuesto), mientras que en la Figura 12 se muestran los motores una vez instalados en sus respectivos soportes.

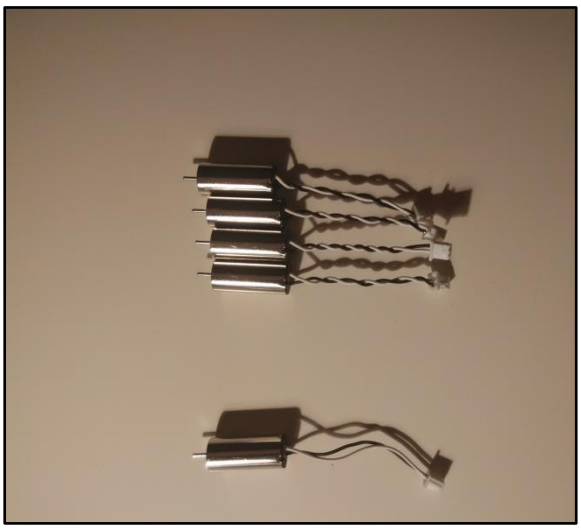


Figura 11 : Montaje CF2 (motores)

Figura 12 : Montaje CF2 (motores y soportes)

En la Figura 13 se muestra el estado de montaje tras instalar toda la planta propulsora (soportes, motores y propulsores) en la placa base y realizar el conexionado. Asimismo, en la Figura 14 se muestra la almohadilla ya adherida a la placa base y la batería antes de su instalación.

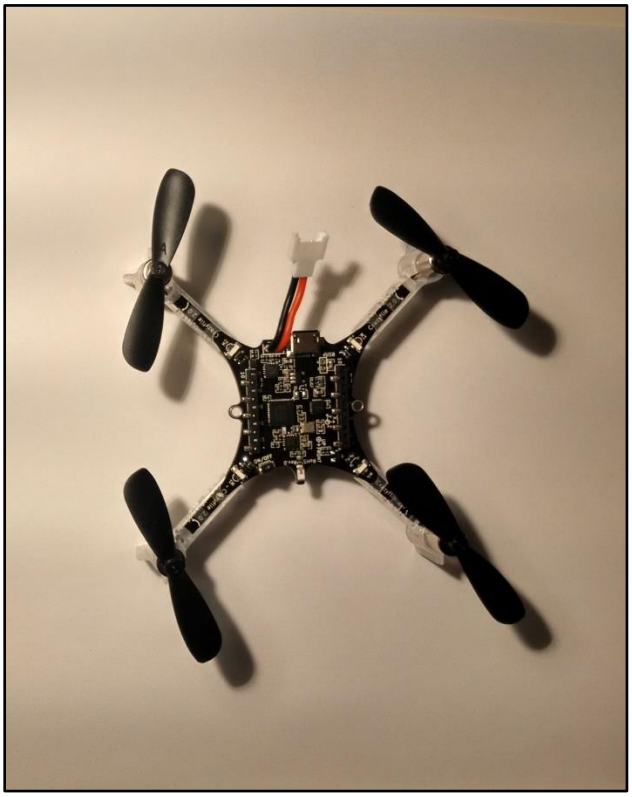


Figura 13 : Montaje CF2 (placa con planta propulsora)

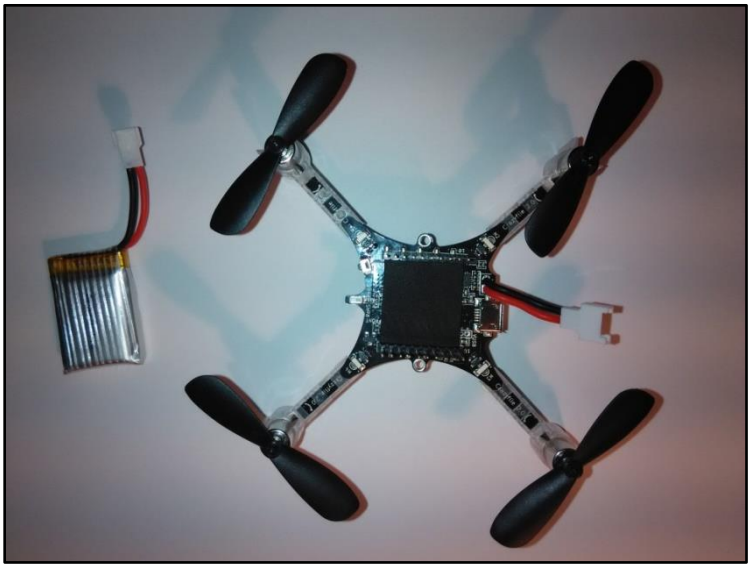


Figura 14 : Montaje CF2 (almohadilla y batería)

Finalmente, en la Figura 15 se muestra el Crazyflie 2.0 una vez finalizado el montaje.

Figura 15 : Montaje CF2 (vehículo completado)

4.3.4 Instalación del software

Siguiendo las instrucciones del fabricante del UAV [41], se instaló en el ordenador portátil que se ha usado como estación de tierra (véase apartado 4.2.6) una plataforma de virtualización (virtual box) [42] para a continuación instalar en la misma una máquina virtual proporcionada por el mismo fabricante (Bitcraze virtual machine) [43]. La máquina virtual empleada ha sido la versión 2016.06 de la misma, que contiene preinstalado todo el software necesario para la gestión del Crazyflie 2.0. La estación de tierra se ejecuta en la máquina virtual, y a través de la antena de la estación de tierra se realizan diversos procesos de gestión del vehículo sin necesidad de conectarlo a la misma, como pueden ser la actualización de firmware o la captura de los datos de vuelo por telemetría. La primera tarea que se realizó, una vez instalada la estación de tierra en la máquina virtual, fue la comprobación de su correcto funcionamiento y la conexión del Crazyflie 2.0 a la misma. Una vez conectado satisfactoriamente se realizó la actualización del firmware de los controladores del vehículo a la versión descrita en el .apartado 7.2.1, que es la que se ha empleado a lo largo del presente trabajo. El proceso fue satisfactorio, comprobándose mediante unos primeros vuelos de verificación que tanto la estación de tierra como el vehículo con el firmware instalado funcionaban correctamente.

5 ESPECIFICACIÓN DEL PROBLEMA A RESOLVER

5.1 Descripción

El problema físico que se trata de resolver mediante simulación por ordenador es el control de la localización de un vehículo aéreo móvil, en términos de posición y orientación, a lo largo del tiempo. Este no es un problema novedoso, ya que se trata, en definitiva, del problema clásico del movimiento de un sólido rígido en un espacio tridimensional y de las técnicas para su control que ya fue resuelto por la ingeniería aeronáutica hace muchos años.

En los últimos años, numerosos grupos de investigación han trabajado en los distintos aspectos relacionados con el control de cuadrirotores, ya que este tipo de vehículos aéreos se vienen usando de forma masiva tanto en investigación como de forma comercial, tal y como se puede comprobar en la bibliografía [9][44] [45] [46] [47][48] [49][50] [51] [52] [53][54] [55] [56] [57] [58] [59][60] [61] [62] [63][64] [65] [66] [67] [68] [69] [70] [71].

Los cuadrirotores son vehículos subactuados, ya que se mueven en un entorno, el espacio 3D, que se caracteriza por tener seis grados de libertad, mientras que cuentan únicamente con cuatro actuadores. La simplificación en el número y disposición de los actuadores se ve contrarrestada por la mayor complejidad en su movilidad y maniobrabilidad, siendo el control de trayectorias más complicado ya que, dependiendo de la posición de los puntos de partida y de llegada, el vehículo tiene que realizar una serie de maniobras auxiliares para alcanzar el punto final.

Los principios físicos a emplear para el estudio del control del movimiento del vehículo en el aire son ampliamente conocidos, y han sido empleados con éxito en la ingeniería aeroespacial para todo tipo de aeronaves. Se basan en las ecuaciones de la mecánica clásica y de la aerodinámica.

Durante la pasada década numerosos grupos de investigación han trabajado en distintos aspectos de este problema [44] [61] [53] [46] [72] [73] [17] [74] [75] [76] [15] [10] [77] [78] [79] [49] [80] [81] [63] [62] [82] [83] [68]. Para ello cada grupo ha desarrollado modelos matemáticos de los multirotores con los que se realizaban experimentos y han diseñado un amplio abanico de sistemas de control de los mismos. La literatura al respecto es abundante, pero presenta la dificultad de una falta de uso homogéneo de la notación y el hecho, si cabe más problemático, de que cada modelo establece sus propios sistemas de referencia, su propia herramienta matemática para describir la orientación y sus propias simplificaciones en lo que respecta a los aspectos físicos que se consideran principales en el modelado matemático. Asimismo, los vehículos empleados y la obtención de los parámetros de los mismos tampoco ha sido homogénea, empleándose tanto vehículos de tamaño y peso reducidos, como el del presente trabajo, que presentan mayor agilidad y maniobrabilidad, como otros de pesos mayores, en los que se introduce un número mayor de sensores, e incluso herramientas de manipulación de objetos. Como consecuencia de esta diversidad, en la actualidad no se considera que exista un modelo matemático único capaz de describir la dinámica de vuelo de cualquier multirotor. Por tanto, cada estudio debe comenzar por establecer su propio modelo matemático de la dinámica del vehículo como paso inicial necesario antes de afrontar el diseño de un sistema de control o la simulación por ordenador de la dinámica. En este trabajo se ha seguido una línea de acción similar a la de otros trabajos de investigación [44] [62] [52] [53] [60] [58] [57] [68] [83] en la

que se ha realizado la deducción del modelo matemático a partir de principios fundamentales para, después, continuar con otros aspectos.

5.2 Análisis previo

Antes de comenzar los desarrollos necesarios para resolver el problema de control planteado se ha realizado un esquema de los bloques conceptuales implicados en el mismo. A pesar de que en el desarrollo se ha optado por emplear una subdivisión del problema diferente, en la fase conceptual de análisis previo fue útil dicho esquema que se muestra en la Figura 16.

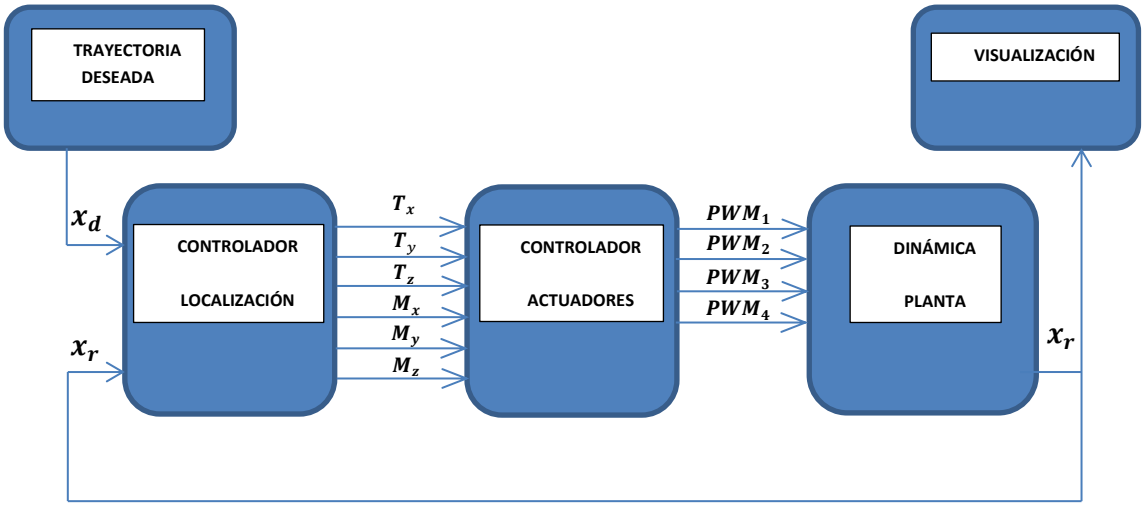


Figura 16 : Esquema conceptual del problema de control a resolver

En el esquema conceptual se muestran los siguientes bloques, con sus respectivas entradas y salidas, que se consideran básicos para la resolución del problema:

- Controlador de localización: se trata del controlador de la dinámica del vehículo. Recibe como entrada el vector de estado actual del vehículo y la trayectoria que deseamos y calcula la fuerza y momento resultante que necesitamos aplicar al mismo para conseguir la trayectoria deseada, mostrándolos a la salida. Su modelo se describe en el capítulo 7 y su simulación en el capítulo 8.
- Controlador de los actuadores: se trata del controlador de los cuatro motores que accionan las hélices. Recibe como entrada la fuerza y momento a aplicar al vehículo y calcula las velocidades de giro de los cuatro actuadores que las van a generar, teniendo en cuenta la dinámica de los mismos. Finalmente muestra en la salida la señal de control en forma de voltaje con modulación por ancho de pulsos. Su modelo se describe en el capítulo 7 y su simulación en el capítulo 8.
- Dinámica de la Planta: es el bloque que conceptualiza el modelo dinámico del Crazyflie 2.0, incluidos motores y hélices. Recibe como entrada la señal de voltaje de control y evoluciona dinámicamente mostrando como salida el vector de estado del vehículo que incluye tanto el vector de configuración (posición y orientación) como su derivada temporal. Su modelo se describe en el capítulo 6 y su simulación en el capítulo 8.

Asimismo, se puede apreciar en la Figura 16 la presencia de dos bloques de interface:

- Trayectoria deseada: es el bloque en el que introducimos los valores de referencia del estado del vehículo, como puede ser la posición que queremos que alcance o la trayectoria que queremos que siga el vehículo.
- Visualización: es un bloque que nos muestra la localización espacial del vehículo a lo largo del tiempo.

6 MODELO DINÁMICO DE LA PLANTA

6.1 Planteamiento

Para crear el modelo matemático de la planta vamos a considerar que el Crazyflie 2.0 consta de dos elementos diferenciados: la aeronave y la planta propulsora. La aeronave está compuesta del circuito impreso que contiene todos los componentes, incluida la batería, y, asimismo, también incluye la planta propulsora a bordo. A su vez, la planta propulsora la consideraremos compuesta de dos subsistemas: el motor y el propulsor. El elemento motor consta de los cuatro actuadores eléctricos que incorpora el vehículo y que transforman la energía eléctrica de la batería en energía mecánica de rotación en sus respectivos ejes, mientras que el elemento propulsor se refiere a las cuatro hélices de paso fijo que actúan como rotores del vehículo aéreo y que le proporcionan la posibilidad de desplazarse a través del aire.

Esta división en función de componentes principales, o productos como se denomina formalmente a los mismos, es la habitual en la industria aeronáutica, donde cada uno de ellos es desarrollado y diseñado por compañías diferentes. Asimismo, se ha considerado que dividiendo el sistema en los tres bloques mencionados (aeronave, motor y propulsor) se presenta de una forma más clara la dinámica global de la planta, ya que cada uno de dichos bloques individuales presenta aspectos dinámicos claramente específicos.

En definitiva, la estrategia adoptada en este estudio para establecer el modelo dinámico de la planta (Crazyflie 2.0) ha sido su división en tres bloques separados que se interconectan en serie y para los se han deducido sus respectivos modelos dinámicos a partir de principios fundamentales de la física. En la Tabla 10 se muestran las características principales de los tres bloques planteados. En dicha tabla se puede confirmar que se trata de tres subsistemas acoplados en serie, ya que la salida de uno es la entrada del siguiente.

Tabla 10 : Bloques integrantes de la planta

Bloque	Modelo	Fundamento	Entrada	Salida
Motor	Eléctrico	Teoría circuitos y máquinas eléctricas	PWM (4)	Giro eje (4)
Propulsor	Aerodinámico	Teoría elemento de pala	Giro eje (4)	-Fuerza aerodinámica -Momento aerodinámico
Aeronave	Mecánico	Ecuaciones Newton-Euler	-Fuerza aerodinámica -Momento aerodinámico	-Posición -Orientación

La planta tiene como entrada los voltajes suministrados a los motores por el microcontrolador principal del Crazyflie 2.0 (STM32F405) en forma de PWM y como salida la posición y orientación de la misma, así como sus derivadas temporales, como pueden ser velocidades y aceleraciones, tanto lineales como angulares. El modelo planteado para la planta trata de reproducir la dinámica de la misma asumiendo las mismas entradas y salidas y tomando los fundamentos físicos más relevantes de cada uno de los subsistemas para establecer el modelo matemático.

Así, la deducción de un modelo matemático razonable del sistema se puede hacer de forma sencilla a partir de leyes físicas, tales como las mencionadas en la Tabla 10. En el apartado 8.7 se realiza la validación del modelo y se justifica la validez de la deducción realizada.

Tal y como se ha comentado anteriormente, el modelo dinámico de los cuadrirotores, y en general el de los multirotores, ha sido ampliamente estudiado por varios grupos de investigación anteriormente [44] [60] [57] [59] [83], por lo que en lo que sigue únicamente se recogerán los aspectos más destacados del caso que se ha estudiado, haciéndose referencia a la amplia literatura existente para obtener información más detallada.

6.2 Hipótesis, simplificaciones, notación y unidades

Para obtener un modelo simplificado del sistema que se pretende simular se deben, en primer lugar, establecer una serie de hipótesis respecto al mismo. Además, se deben realizar ciertas simplificaciones que consideren despreciables los aspectos secundarios de la física que gobierna el sistema con objeto de retener en el modelo únicamente los aspectos principales, que son los que realmente influyen de forma significativa en las características dinámicas del sistema. Estos dos procesos (establecer hipótesis e introducir simplificaciones) son empleados de forma rutinaria en la elaboración de cualquier modelo matemático de la dinámica de un sistema, donde hay que encontrar una solución de compromiso entre la precisión del modelo y su simplicidad. Se detallan a continuación las principales hipótesis y simplificaciones introducidas, así como la notación empleada en el modelo. Tal y como ya se ha mencionado, será la validación del modelo detallada en el apartado 9.3, la que proporcione una confirmación del acierto de las decisiones tomadas.

6.2.1 Hipótesis

En la deducción del modelo se ha considerado que:

- El vehículo es un sólido rígido. Esta hipótesis es razonable dado que los materiales de los que está fabricada la estructura del vehículo tienen unas características de resistencia y rigidez suficientes teniendo en cuenta las fuerzas a las que van a estar sometidos.
- El vehículo es completamente simétrico, tanto desde el punto de vista geométrico como respecto a la geometría de masas. La hipótesis es razonable, aunque en realidad el vehículo presenta pequeñas faltas de simetría geométrica en la tarjeta impresa, al igual que pequeñas asimetrías en la geometría de masas debidas a los distintos pesos de los componentes electrónicos discretos instalados en la placa (ver Figura 6, Figura 7). No obstante, las asimetrías geométricas son pequeñas en comparación con las dimensiones del vehículo y las asimetrías en la geometría de masas son también pequeñas en comparación con los mayores contribuyentes a la misma que son la batería y los actuadores (ver Tabla 2), que están en posiciones simétricas. Una de las consecuencias de esta hipótesis es que el centro de masas se considera coincidente con el centro de simetría del vehículo
- Los ejes de rotación de los cuatro rotores son paralelos al eje de simetría vertical del vehículo. Esta hipótesis depende de realizar un correcto montaje de la aeronave (ver apartado 4.3.3) manteniendo la adecuada alineación de los motores en sus receptáculos de plástico. Las piezas en las que van instalados los motores se han

diseñado con un ajuste pequeño que, en términos generales, asegura el cumplimiento de esta hipótesis.

6.2.2 Simplificaciones

Los siguientes aspectos físicos se han considerado despreciables y no se han tenido en cuenta en el modelo:

- Los efectos aerodinámicos secundarios. Una descripción de los mismos y de los motivos para no introducirlos en el modelo se incluye en el apartado 6.8.
- Los momentos de precesión giroscópica generados por el giro de los rotores. Son momentos pequeños en comparación con los momentos debidos a la aerodinámica, debido al bajo valor del momento de inercia de las hélices.

6.2.3 Notación y unidades

La notación respecto a los parámetros importantes de los micro vehículos aéreos no tripulados no está, hasta la fecha, unificada, empleando cada autor la que le parece más oportuna. En el presente trabajo se han tenido en cuenta las siguientes consideraciones:

- Se ha empleado un punto para separar la parte entera y decimal de los números. Esto facilita la compatibilidad con el código y evita errores al transcribir los valores de los parámetros. Esta notación está aceptada por el BIPM [84].
- Se ha empleado un punto encima de una variable para referirse a su primera derivada temporal, y dos puntos para su segunda derivada temporal.
- Se ha empleado un subíndice de una letra o número para referirse al sistema de referencia o eje en el que las coordenadas de la variable están expresadas, o bien para caracterizar mejor una constante.
- Se ha empleado un subíndice con dos letras en las matrices de transformación para expresar que transforman las coordenadas expresadas en el sistema de referencia de la segunda letra a coordenadas expresadas en el sistema de referencia de la primera letra. Asimismo, en el caso de la matriz de inercia se ha usado la notación habitual para los momentos y productos de inercia que se refiere a los ejes correspondientes.

Se han empleado unidades del sistema internacional en todo el trabajo.

6.3 Sistemas de referencia

Se han empleado dos sistemas de referencia ortogonales y a derechas para poder establecer las ecuaciones del modelo matemático tal y como se muestra en la Figura 17.

En primer lugar se cuenta con un sistema de referencia inercial E ≡ $(O_E X_E Y_E Z_E)$, identificado como sistema de referencia del laboratorio. El eje $O_E Z_E$ se considera perpendicular al suelo y los ejes $O_E X_E$ y $O_E Y_E$ coincidentes con el plano del suelo.

Asimismo, se ha establecido un sistema de referencia no inercial ligado al vehículo B ≡ $(O_B X_B Y_B Z_B)$. El sistema de referencia ligado tiene su origen en el centro de masas y sus ejes son ejes principales de inercia del sólido rígido, debido a la simetría del vehículo. El sistema ligado empleado es el que emplea el firmware del controlador del vehículo, y que es diferente al empleado por los sensores y al habitual en aeronáutica, tal y como se detalló en el apartado 4.2.2.

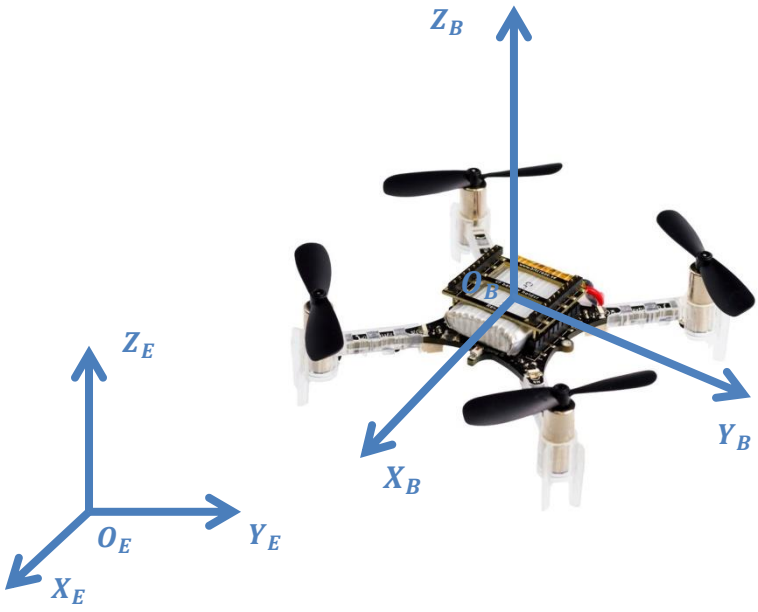


Figura 17 : Sistemas de referencia

En la Figura 18 se muestra una vista superior del dron en la que se puede apreciar el criterio de signos empleado para las rotaciones del vehículo alrededor de los ejes de referencia del sistema ligado, así como los sentidos de giro de los actuadores establecidos por el fabricante. Se puede observar que la dirección de avance seleccionada por el fabricante, y mostrada por una pequeña protuberancia del circuito impreso, implica una configuración en X (cross configuration), en la que, a diferencia de la más habitual configuración en cruz (+ configuration), todos los actuadores contribuyen en la generación del momento de giro en cualquiera de los ejes. Esta contribución se visualiza claramente en la matriz de asignación Γ descrita en la ecuación (13) del apartado 6.8 y en la ecuación de distribución del controlador de la planta de potencia, ecuación (26) del apartado 7.4, en las que podemos observar la contribución de todos los rotores en los distintos componentes del momento resultante.

La principal diferencia entre la configuración en cruz y la configuración en X consiste en que en la configuración en cruz los actuadores se sitúan en los ejes de referencia, permitiendo que se varíe el ángulo de alabeo o el de balanceo actuando únicamente en las velocidades de rotación de los dos actuadores situados en el otro eje , mientras que se mantienen las revoluciones del resto de actuadores en el mismo valor, mientras que en la configuración en X ninguno de los actuadores está situado en los ejes de referencia, por lo que para la variación de dichos ángulos se hace necesario en todos los casos actuar sobre la velocidad de giro de los cuatro actuadores. En lo que respecta a la variación del ángulo de guiñada, así como a los movimientos de subida y bajada, ambas configuraciones son similares. El control de ambas configuraciones es básicamente el mismo, con las diferencias en la matriz de asignación Γ antes mencionadas. La maniobrabilidad de la configuración en X es ligeramente mayor en comparación a la configuración en cruz, para el mismo tipo de movimiento deseado [85].

Los sentidos de giro de los rotores mostrados en la Figura 18 son importantes para establecer el sentido de los momentos y su contribución al momento resultante, nuevamente a través de la matriz de asignación Γ descrita en 6.8.

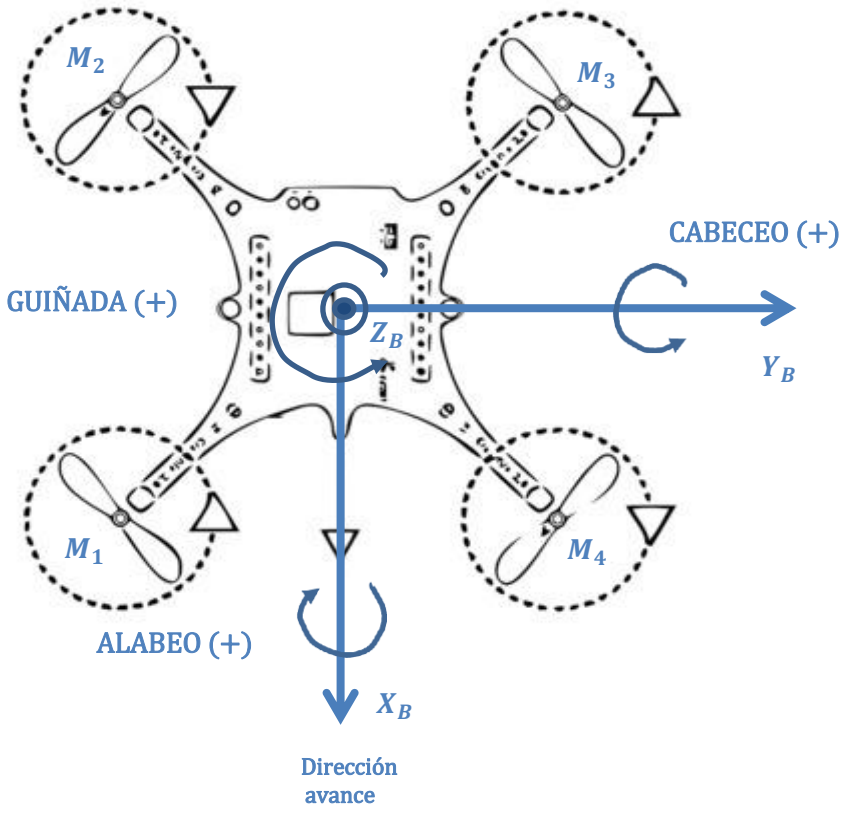


Figura 18 : Sentido giro motores y criterio signos giros vehículo (vista superior)

6.4 Localización espacial

Para la localización espacial del Crazyflie 2.0 se han empleado seis coordenadas, tres para definir la posición y otras tres para definir la orientación espacial del vehículo.

La posición del vehículo se establece mediante las coordenadas cartesianas del centro de masas respecto al sistema inercial del laboratorio, tal y como se muestra en la ecuación (3).

$$\zeta_E = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{3}$$

Para definir la orientación del vehículo se emplean ángulos de Euler de giñada, cabeceo y alabeo, adoptando el denominado convenio ZYX, tal y como se muestra en la ecuación (4). Es decir, la orientación del sistema de referencia móvil se obtiene a partir de un sistema de referencia coincidente con el sistema de referencia del laboratorio al que hacemos girar un

ángulo ψ alrededor del eje Z_E, continuando con un giro de ángulo θ alrededor del nuevo eje Y', para finalizar con un giro de un ángulo ϕ en torno al nuevo eje X'. Tras estos giros el sistema de referencia es coincidente con el sistema de referencia ligado al vehículo que se ha definido en 6.3. Otra forma equivalente de representar dicha orientación, que proporciona los mismos resultados de la ecuación (4), es considerar giros respecto al sistema de referencia inercial ligado al laboratorio, en vez de respecto a un sistema de referencia ligado al vehículo. Así, si se realiza primero un giro de un ángulo de alabeo ϕ en torno al eje X_E, seguido de un giro de un ángulo de cabeceo θ alrededor del eje Y_E, para finalizar con un ángulo de guiñada ψ alrededor del eje Z_E, la orientación queda definida de forma equivalente. En el primer caso se denomina al criterio empleado convenio de Euler (o Tait-Bryan) ZYX, y en el segundo convenio alabeo-cabeceo-guiñada (o roll-pitch-yaw).

$$\mathbf{\Phi} = \begin{pmatrix} \phi \\ \theta \\ \psi \end{pmatrix} \tag{4}$$

6.5 Parámetros y variables del modelo de la planta

El modelo de la planta emplea una serie de parámetros tanto mecánicos, como aerodinámicos y eléctricos. Asimismo el modelo se define en base un cierto número de variables, tanto independientes como dependientes. En este apartado se enumeran los parámetros y variables que se han empleado, mencionando la notación elegida y, en el caso de los parámetros incluyendo también información acerca del origen del valor de los mismos.

A la hora de elegir los parámetros que se han empleado en el modelo se ha realizado una revisión bibliográfica para decidir si era necesario o no llevar a cabo experimentos de identificación de parámetros [86] para su determinación. Dado que se han encontrado estudios recientes realizados por grupos de investigación con el mismo vehículo en los que se obtenían los parámetros principales necesarios para el modelo [4] [9] mediante experimentación, se ha optado por usar dichos valores y considerar posteriormente, a la hora de realizar la validación del modelo si dicha decisión ha sido acertada o no.

6.5.1 Parámetros mecánicos

Se trata de los parámetros geométricos del vehículo que son relevantes para el modelo, así como su masa, su matriz de inercia y la aceleración de la gravedad. Los valores que se han empleado se muestran en la Tabla 11.

Tabla 11 : Parámetros mecánicos

Parámetro	Notación	Valor	
Distancia eje actuador-eje X_B (Y_B)	$d = \frac{\sqrt{2}}{2} d'(*)$	$3.97 \cdot 10^{-2}$	m
Masa	m	$2.70 \cdot 10^{-2}$	kg
Aceleración gravedad	g	9.81	$m \cdot s^{-2}$
Momento inercia eje X_B	I_{xx}	$1.66 \cdot 10^{-5}$	$kg \cdot m^2$
Momento inercia eje Y_B	I_{yy}	$1.66 \cdot 10^{-5}$	$kg \cdot m^2$
Momento inercia eje Z_B	I_{zz}	$2.93 \cdot 10^{-5}$	$kg \cdot m^2$

(*) d' es la distancia entre el centro geométrico del vehículo y el eje de cualquiera de los actuadores

El único parámetro geométrico que se usa en el modelo es d, la distancia entre el eje de un actuador y el eje X_B, o bien el eje Y_B. Dada la simetría del diseño ambas distancias son iguales y, además lo son para los cuatro actuadores. Esta distancia es necesaria para calcular la contribución de las fuerzas de tracción generadas por las hélices a los momentos de cabeceo y alabeo. En la Figura 19 se muestra dicha distancia (d) y se observa que para su obtención basta multiplicar la distancia entre el centro geométrico del vehículo y el eje del rotor (d') por el coseno o el seno de 45º. La distancia d' se ha tomado de las especificaciones del vehículo[1].

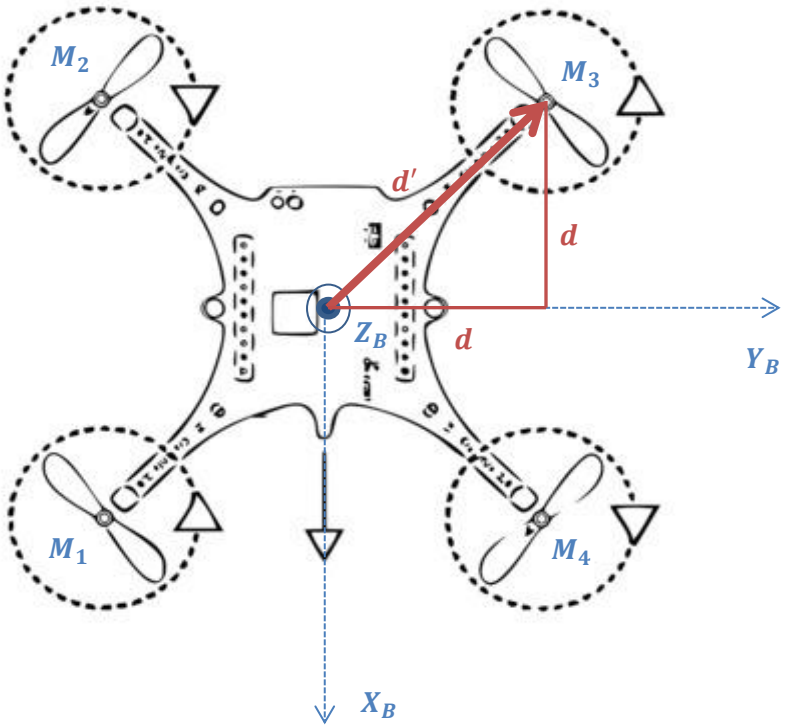


Figura 19 : Parámetros geométricos

La masa, de la misma forma, se toma directamente de las especificaciones del vehículo. En cuanto a la matriz de inercia se hace notar que teniendo en cuenta que los ejes del sistema de referencia elegido en 6.3 son ejes principales de inercia, la matriz de inercia será una matriz diagonal. Los valores de los momentos de inercia se han tomado de los resultados de un experimento reciente realizado sobre el vehículo en el que se empleaba el principio del periodo del péndulo para calcularlos [4].

6.5.2 Parámetros aerodinámicos

Para los cálculos aerodinámicos se necesita relacionar las fuerzas y momentos generados por los rotores con sus velocidades de giro. Dado que en el modelo solo se ha tenido en cuenta la aerodinámica primaria, con las simplificaciones descritas en el apartado 6.8, y que se consideran despreciables los efectos aerodinámicos secundarios, solo se precisa de un número reducido de parámetros aerodinámicos. Los parámetros son del tipo agrupado (lumped) dado que de esta forma se simplifica la caracterización del fenómeno [56]. En la Tabla 12 se muestran los valores de los coeficientes de tracción y de momento de las hélices. Dichos valores se han tomado de los resultados de una serie de experimentos recientes realizados sobre el vehículo [4], en los que se conseguían obtener los parámetros relevantes de los mapeados entre aerodinámica y velocidades de giro de los rotores.

Tabla 12 : Parámetros aerodinámicos

Parámetro	Notación	Valor
Coeficiente de tracción cuadrático	C_{t1}	$1.28 \cdot 10^{-8}$ $N \cdot s^2$
Coeficiente de tracción lineal	C_{t0}	$1.56 \cdot 10^{-5}$ $N \cdot s$
Coeficiente de momento cuadrático	C_{m1}	$7{,}65 \cdot 10^{-11}$ $N \cdot m \cdot s^2$
Coeficiente de momento lineal	C_{m0}	$9{,}28 \cdot 10^{-8}$ $N \cdot m \cdot s$

6.5.3 Parámetros eléctricos

En el caso de los motores eléctricos, se ha precisado de dos parámetros para caracterizar la dinámica de los mismos. Sus valores se han tomado de los resultados de un experimento reciente de identificación de parámetros realizado sobre el vehículo [23], realizando las necesarias conversiones. En la Tabla 13 se muestran los valores de la constante ganancia y la contante de tiempo.

Tabla 13 : Parámetros eléctricos

Parámetro	Notación	Valor
Ganancia	K_m	$2.37 \cdot 10^{-6}$ V^{-1}
Constante tiempo	T_m	$6.46 \cdot 10^{-2}$ s

6.5.4 Variable independiente

Tal y como ocurre en todos los problemas de control que tratan con sistemas dinámicos, la variable independiente a considerar es el tiempo.

6.5.5 Variables dependientes

En la Tabla 14 se listan las principales variables dependientes que se han empleado al establecer el modelo del Crazyflie 2.0. Se ha empleado el subíndice para referirse al sistema de referencia en el que se expresa la variable. Por comodidad se ha elegido expresar las variables referentes a la translación en el sistema de referencia del laboratorio (E), mientras que las

variables de rotación se han expresado en el sistema de referencia ligado al vehículo (B). Esta es una elección habitual en este tipo de modelos.

Tabla 14 : Variables empleadas en el modelo

Translación		Rotación	
$\boldsymbol{F}_E = \begin{pmatrix} F_x \\ F_y \\ F_z \end{pmatrix}$	Fuerza resultante aplicada	$\boldsymbol{M}_B = \begin{pmatrix} M_x \\ M_y \\ M_z \end{pmatrix}$	Momento resultante aplicado
$\boldsymbol{a}_E = \begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix}$	Aceleración lineal centro masas	$\boldsymbol{\alpha}_B = \begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix}$	Aceleración angular vehículo
$\boldsymbol{v}_E = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix}$	Velocidad lineal centro masas	$\boldsymbol{\omega}_B = \begin{pmatrix} p \\ q \\ r \end{pmatrix}$	Velocidad angular vehículo
$\boldsymbol{\zeta}_E = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$	Posición centro masas	$\dot{\boldsymbol{\Phi}} = \begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix}$	Derivada temporal ángulos de Euler (Tait-Bryan)
		$\boldsymbol{\Phi} = \begin{pmatrix} \phi \\ \theta \\ \psi \end{pmatrix}$	Ángulos de Euler (Tait-Bryan)
F_3	Fuerza aerodinámica eje Z_B	M_1	Momento aerodinámico eje X_B
		M_2	Momento aerodinámico eje Y_B
		M_3	Momento aerodinámico eje Z_B
		ω_i	Velocidad rotación eje actuadores
		V_i	Voltaje de mando de los actuadores
$\boldsymbol{X} = (x \quad y \quad z \quad \phi \quad \theta \quad \psi \quad \dot{x} \quad \dot{y} \quad \dot{z} \quad \dot{\phi} \quad \dot{\theta} \quad \dot{\psi})^T$		Vector de estado	

6.6 Matrices de rotación y transformación

Para establecer correctamente el modelo de la planta es necesario determinar las relaciones entre la orientación de los sistemas de referencia que hemos definido, así como la relación entre la velocidad angular del vehículo y la variación con el tiempo de los ángulos de Euler seleccionados.

Los dos sistemas de referencia están relacionados mediante una matriz de rotación ortogonal del grupo especial euclídeo SE(3). Se obtiene como producto de las tres rotaciones individuales mencionadas en el apartado 6.4, tal y como se muestra en la ecuación (5).

$$\boldsymbol{R}_{EB} = \boldsymbol{R}_z(\psi) \cdot \boldsymbol{R}_y(\theta) \cdot \boldsymbol{R}_x(\phi) = \begin{pmatrix} cos\psi cos\theta & cos\psi sen\theta sen\phi - sen\psi cos\phi & sen\psi sen\phi + cos\psi sen\theta cos\phi \\ sen\psi cos\theta & cos\psi cos\phi + sen\psi sen\theta sen\phi & sen\psi sen\theta cos\phi - cos\psi sen\phi \\ -sen\theta & cos\theta sen\phi & cos\theta cos\phi \end{pmatrix} \quad (5)$$

La variación temporal de los ángulos de Euler está relacionada con la velocidad angular del vehículo a través de la matriz de transformación $\boldsymbol{W}_{EB}$. Dicha matriz se obtiene por inversión de la matriz $\boldsymbol{W}_{BE}$ que, a su vez se puede obtener inicialmente aplicando las rotaciones elementales necesarias a las variaciones temporales de los ángulos de Euler. En la ecuación (6) se muestra el resultado final de la matriz de transformación.

$$\dot{\boldsymbol{\Phi}} = \boldsymbol{W}_{EB} \cdot \boldsymbol{\omega}_B$$
$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 1 & sen\phi tan\theta & cos\phi tan\theta \\ 0 & cos\phi & -sen\phi \\ 0 & sen\phi sec\theta & cos\phi sec\theta \end{pmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix} \quad (6)$$

Los detalles de la deducción de las matrices $\boldsymbol{R}_{EB}$ y $\boldsymbol{W}_{EB}$ se ha incluido en el ANEXO 1.

6.7 Modelo eléctrico de los motores

Cada uno de los cuatro motores eléctricos que realizan la función de actuadores en el vehículo objeto de estudio recibe la señal de mando, en forma de voltaje modulado por ancho de pulsos (PWM), del microcontrolador principal, y proporciona como salida el giro de su eje a unas revoluciones determinadas. Los motores que incorpora el vehículo son motores de corriente continua de imán permanente de reducidas dimensiones. Dichos motores, tal y como se ha descrito en el apartado 4.2.2, presentan una dinámica extremadamente rápida y son capaces de responder a una señal de mando con tiempos muy inferiores a los característicos de la dinámica del vehículo, por lo que incluso se podría considerar dicha respuesta instantánea. No obstante, se ha optado por plantear un modelo dinámico teniendo en cuenta los fundamentos físicos de su funcionamiento.

Las ecuaciones (7) expresan la ley de tensiones de Kirchhoff y la fuerza contraelectromotriz generada por el giro del motor. Ambas gobiernan la dinámica de los motores de nuestro caso.

$$V_i = V_b + RI + L\dot{I}$$
$$\dot{\omega}_i = \frac{V_b}{k_b} \quad (7)$$

donde:

V_i	tensión de mando
I	intensidad circuito
V_b	fuerza contraelectromotriz inducida
R, L	resistencia e inducción circuito
ω_i	velocidad de rotación eje motor
k_b	constante del motor

Una vez aplicadas las necesarias agrupaciones y simplificaciones, y una vez aplicada la transformada de Laplace a las ecuaciones (7) se obtiene la ecuación (8), que describe la dinámica de los motores.

$$\frac{\omega_i(s)}{V_i\,(s)} = \frac{K_m}{T_m s + 1} \quad (8)$$

donde:

V_i	tensión de mando
ω_i	velocidad de rotación eje motor
K_m	constante de ganancia del motor
T_m	constante de tiempo del motor

La ecuación (8) es la ecuación que gobierna el bloque motor definido en 6.1 y es la que se ha empleado en la simulación de dicho bloque. En la Figura 20 se muestra la respuesta de uno de los motores, simulada en MATLAB®, a una entrada escalón.

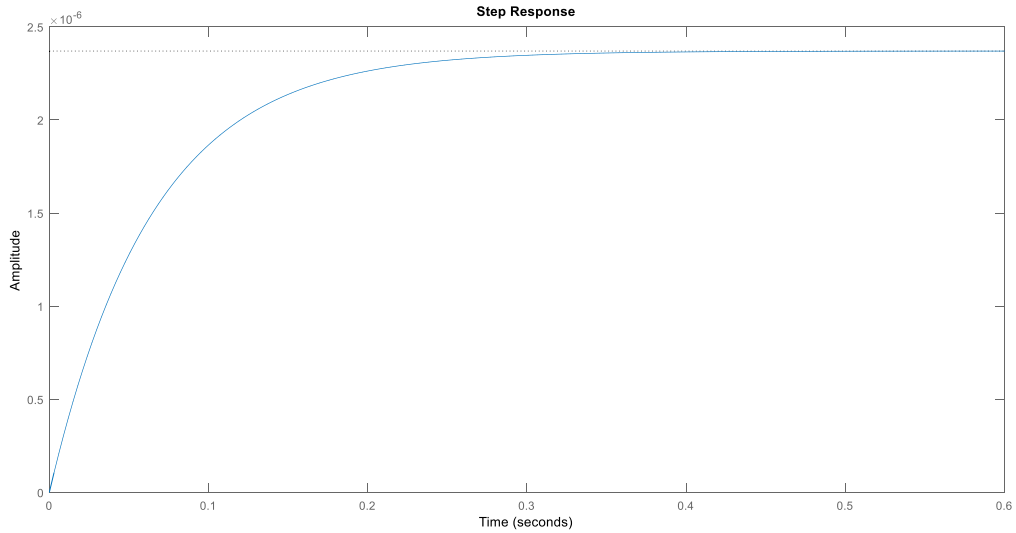


Figura 20 : Respuesta temporal actuadores

Tanto los estudios teóricos [87] [88] como los experimentos de identificación de sistemas llevados a cabo [4] [44] [52] confirman que una dinámica de primer orden es una buena aproximación para dichos motores en este tipo de aplicaciones.

6.8 Modelo aerodinámico de las hélices

El Crazyflie 2.0 consta, al igual que todos los cuatrirotores, de cuatro hélices o rotores que giran para crear las fuerzas y momentos necesarios para el movimiento del vehículo en el aire. El modelo aerodinámico debe tener en cuenta tanto dichas fuerzas y momentos generados por los rotores como las fuerzas y momentos originados por el movimiento del vehículo en el aire. Los efectos aerodinámicos se pueden clasificar en dos grandes grupos: primarios y secundarios, tal y como se resume en la Tabla 15.

Tabla 15 : Efectos aerodinámicos

Primarios	**Secundarios**
Tracción vertical	Bataneo de las palas
Momento resistente	Resistencia parásita de las palas
	Resistencia aerodinámica del vehículo
	Efecto suelo

A pesar de que algún estudio [62] ha identificado la necesidad de tener en cuenta los efectos secundarios en el modelo aerodinámico de un cuatrirotor para obtener modelos más fieles a la realidad, especialmente el bataneo de las palas de los rotores, es un hecho constatado que la mayoría de los modelos obvian dichos efectos consiguiendo resultados aceptables. Por lo tanto,

en base a la existencia de dichos estudios similares, no se han considerado los efectos secundarios en el modelo. Se incluye a continuación una descripción más detallada de cada uno de estos efectos, tanto primarios como secundarios. Más información sobre los mismos se puede encontrar en [56].

El giro de la hélice en el aire genera de forma primaria tracción en la dirección del eje de giro y un momento resistente de sentido opuesto al del giro que también localizado en el mismo eje. El valor de dicha fuerza de tracción y momento resistente se puede estimar a partir de la teoría del elemento de pala [89]. Las ecuaciones (9) y (10) nos permiten calcular los módulos de dichos vectores.

$$T = \rho(\pi R^2)(\omega_i R)^2 \frac{\sigma_e}{6} C_l \tag{9}$$

$$M = \rho(\pi R^2)(\omega_i R)^2 \frac{\sigma_e}{9} C_d \tag{10}$$

donde:

	T	tracción generada por la hélice
	M	par resistente de la hélice
	ρ	densidad del aire
	R	radio de la hélice
	ω_i	velocidad angular de la hélice
	σ_e	solidez efectiva de la hélice
	C_l	coeficiente de sustentación del elemento de pala
	C_d	coeficiente de resistencia del elemento de pala

Despreciando las variaciones de densidad durante el vuelo, y teniendo en cuenta que vamos a usar un tipo único de hélices en nuestros experimentos, se pueden emplear parámetros agrupados que contengan todas las magnitudes que van a permanecen constantes durante los experimentos, con objeto de simplificar las fórmulas. Así, en nuestro caso las expresiones (9) y (10) con los coeficientes agrupados se transforman en las expresiones (11) y (12).

$$T = C_{t1}{\omega_i}^2 + C_{t0}\omega_i \tag{11}$$

$$M = C_{m1}{\omega_i}^2 + C_{m0}\omega_i \tag{12}$$

donde:

	T	tracción generada por la hélice
	M	par resistente de la hélice
	ω_i	velocidad angular de la hélice
	$C_{t1,0}$	coeficientes de tracción agrupado
	$C_{m1,0}$	coeficientes de par agrupado

En la Figura 21 se muestran la localización en el Crazyflie 2.0 de las fuerzas y momentos aerodinámicos generados teniendo en cuenta únicamente los efectos aerodinámicos primarios. El valor de los módulos de los vectores es el dado por las fórmulas (11) y (12).

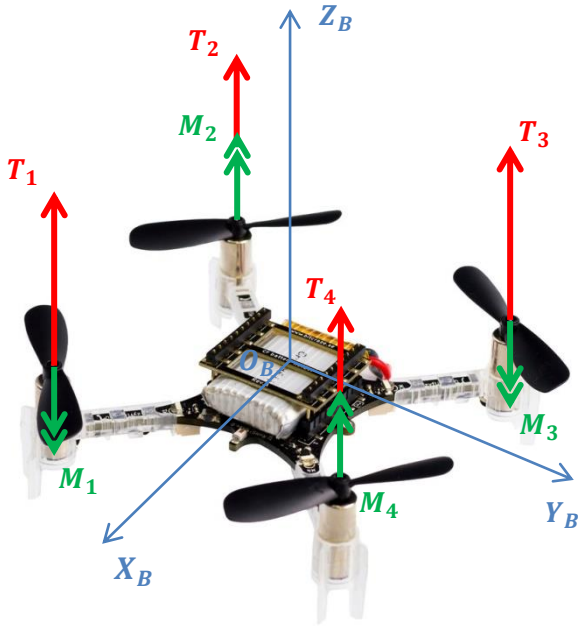


Figura 21 : Fuerzas y momentos aerodinámicos

Las fuerzas y momentos aerodinámicos individuales mostrados en la Figura 21 contribuyen a la fuerza y momento aerodinámico resultantes a través de las matrices de asignación (Γ_1, Γ_0) mostradas en la ecuación (13), que son las matrices que relacionan dichas fuerzas y momentos aerodinámicos resultantes con las velocidades de giro de las hélices. Las matrices de asignación tienen en cuenta tanto las ecuaciones (11) y (12) deducidas anteriormente, como la configuración geométrica del vehículo tanto en lo que respecta a la distancia existente entre los ejes de los actuadores y los ejes del sistema de referencia ligado (véase 6.5.1). Así la fuerza resultante (F_3) está dirigida según el sentido positivo del eje Z_B, mientras que las tres componentes del momento resultante (M_1, M_2, M_3) están dirigidas según los sentidos positivos de los ejes $(X_B Y_B Z_B)$ respectivamente.

$$\begin{pmatrix} F_3 \\ M_1 \\ M_2 \\ M_3 \end{pmatrix} = \Gamma_1 \begin{pmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{pmatrix} + \Gamma_0 \begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{pmatrix} = \tag{13}$$

$$= \begin{pmatrix} C_{t1} & C_{t1} & C_{t1} & C_{t1} \\ -dC_{t1} & -dC_{t1} & dC_{t1} & dC_{t1} \\ -dC_{t1} & dC_{t1} & dC_{t1} & -dC_{t1} \\ -C_{m1} & C_{m1} & -C_{m1} & C_{m1} \end{pmatrix} \begin{pmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{pmatrix} + \begin{pmatrix} C_{t0} & C_{t0} & C_{t0} & C_{t0} \\ -dC_{t0} & -dC_{t0} & dC_{t0} & dC_{t0} \\ -dC_{t0} & dC_{t0} & dC_{t0} & -dC_{t0} \\ -C_{m0} & C_{m0} & -C_{m0} & C_{m0} \end{pmatrix} \begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{pmatrix}$$

La ecuación (13) es la ecuación que gobierna el bloque propulsor definido en 6.1 y es la que se ha empleado en la simulación de dicho bloque. Conviene aclarar que la fuerza y momento resultantes mencionados en ella son solo aerodinámicos, y que el vehículo está sometido, además, a la fuerza de la gravedad que se incluye en el modelo del mecánico del sólido rígido.

Respecto a los efectos aerodinámicos secundarios, se proporcionan a continuación algunos detalles de los mismos aunque, tal y como se ha mencionado anteriormente, no se han tenido en cuenta en el modelo de la planta.

Tanto el bataneo como la resistencia inducida de las palas introducen fuerzas en el plano $X_B Y_B$, que es el que contiene las direcciones subactuadas del vehículo. El bataneo consiste en un fenómeno aeroelástico dinámico en el que intervienen las cargas aerodinámicas generadas por la geometría de los rotores y los esfuerzos y deformaciones del material del que están hechos los rotores. Dado que las cargas externas dependen de la geometría y que esta se modifica como consecuencia de las mismas, nos encontramos ante un fenómeno dinámico que hace que las palas de los rotores puedan flexar y de hecho oscilen en su movimiento de rotación, generando fuerzas horizontales. Asimismo, el hecho de que la resistencia inducida por la pala que avanza sea superior a la que genera la pala que retrocede tiene como resultado, nuevamente una fuerza horizontal. Ambos fenómenos son proporcionales a la velocidad horizontal del vehículo.

Por otro lado, el vehículo se mueve a través del aire y, dado que se trata de un fluido, siempre aparece una fuerza de resistencia al avance, de la misma dirección del movimiento y sentido contrario. Dicha fuerza, como todas las fuerzas aerodinámicas es proporcional al cuadrado de la velocidad de movimiento, por lo que se puede despreciar si las velocidades con las que se desplaza el vehículo son pequeñas.

Finalmente, cuando el vehículo se encuentra próximo al suelo aparece el denominado efecto suelo, ampliamente estudiado en aerodinámica, que incrementa el empuje producido por los rotores como consecuencia de la modificación del campo de velocidades de la corriente ante la presencia de la barrera física que representa el suelo.

6.9 Modelo mecánico del sólido rígido

El modelo mecánico del vehículo relaciona las aceleraciones lineales y angulares del mismo con las fuerzas y momentos aplicados. Se trata, en definitiva, del movimiento de un sólido rígido que tiene 6 grados de libertad, y que vamos a considerar que está sometido únicamente a fuerzas y momentos debidos a la los efectos aerodinámicos primarios y a la fuerza de la gravedad, tal y como se muestra en la Figura 22.

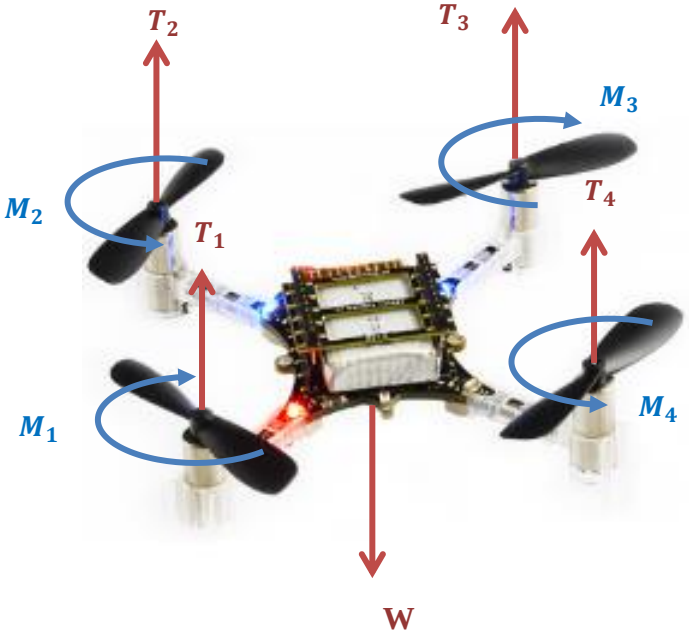


Figura 22 : Fuerzas y momentos aplicados (aerodinámica y gravedad)

Para establecer el modelo matemático a partir de las leyes físicas que gobiernan la dinámica de este subsistema contamos con la segunda ley de Newton para el movimiento de translación y con la ecuación de Euler para la rotación. Asimismo, se deben tener en cuenta las hipótesis y simplificaciones mencionadas en 6.2, y además emplear los sistemas de referencia establecidos en el apartado 6.3. Por conveniencia, ya que se simplifican las ecuaciones, expresaremos las ecuaciones de la translación del sólido en el sistema de referencia inercial del laboratorio (E), mientras que las ecuaciones de la rotación se expresarán en el sistema de referencia no inercial ligado al vehículo (B).

Las ecuaciones de Newton-Euler son las ecuaciones (14) y (15). Podemos observar que la primera de ellas está expresada en el sistema de referencia inercial, por lo que no aparecen fuerzas centrípetas ni de Coriolis. Relaciona la fuerza resultante aplicada con la aceleración lineal del vehículo. La segunda ecuación, expresada en el sistema de referencia no inercial ligado al vehículo, relaciona el momento resultante aplicado al vehículo con su velocidad angular y su aceleración angular, y tiene en cuenta que el sistema de referencia seleccionado está constituido por ejes principales de inercia.

$$\boldsymbol{F}_E = m\,\boldsymbol{a}_E \tag{14}$$

$$\boldsymbol{M}_B = \boldsymbol{I}\,\dot{\boldsymbol{\omega}}_B + \boldsymbol{\omega}_B \times \boldsymbol{I}\,\boldsymbol{\omega}_B \tag{15}$$

A continuación debemos obtener la expresión de la fuerza y momento resultantes. Dado que el modelo únicamente va a considerar la fuerza de la gravedad y las fuerzas y momentos aerodinámicos, las expresiones de las resultantes se pueden expresar directamente a partir de los valores obtenidos en la sección 6.8, con solo introducir como elementos adicionales la gravedad y la matriz de rotación. Esta matriz formaliza el cambio de coordenadas de la fuerza

aerodinámica resultante, del sistema de referencia ligado al vehículo al sistema de referencia del laboratorio. La ecuación (16) muestra la fuerza resultante y la (17) el momento resultante.

$$\boldsymbol{F}_E = \begin{pmatrix} F_x \\ F_y \\ F_z \end{pmatrix} = \boldsymbol{R}_{EB} \begin{pmatrix} 0 \\ 0 \\ F_3 \end{pmatrix} + m \begin{pmatrix} 0 \\ 0 \\ -g \end{pmatrix} \tag{16}$$

$$\boldsymbol{M}_B = \begin{pmatrix} M_x \\ M_y \\ M_z \end{pmatrix} = \begin{pmatrix} M_1 \\ M_2 \\ M_3 \end{pmatrix} \tag{17}$$

Introduciendo las expresiones de la fuerza y momento resultantes en las ecuaciones de Newton-Euler obtenemos el sistema de ecuaciones que gobierna la dinámica del sistema y que se muestra en las expresiones (18) y (19).

$$\begin{aligned} m\ddot{x} &= F_3(cos\psi sen\theta cos\phi + sen\psi sen\phi\ ) \\ m\ddot{y} &= F_3(sen\psi sen\theta sen\phi - cos\psi sen\phi) \\ m\ddot{z} &= F_3(cos\theta cos\phi) - mg \end{aligned} \tag{18}$$

$$\begin{aligned} I_{xx}\dot{p} &= qr\left(I_{yy} - I_{zz}\right) + M_1 \\ I_{yy}\dot{q} &= rp(I_{zz} - I_{xx}) + M_2 \\ I_{zz}\dot{r} &= pq\left(I_{yy} - I_{xx}\right) + M_3 \end{aligned} \tag{19}$$

La simulación por ordenador del vuelo del Crazyflie 2.0 se reduce a la doble integración numérica del conjunto de seis ecuaciones diferenciales de segundo orden no lineales y con acoplamiento que se establecen en las ecuaciones (18) y (19). Para conseguir dicha resolución se necesita contar también con la ecuación (6) para relacionar la velocidad angular del vehículo con el cambio de su orientación y las ecuaciones (8) y (13) para tener en cuenta los aspectos relacionados con la planta motopropulsora.

Se puede observar que la dinámica de rotación sólo depende de los momentos aerodinámicos y del estado de rotación del vehículo, mientras que la translación depende tanto de las fuerzas aerodinámicas y el estado de translación del vehículo como, además, del estado de rotación del mismo, debido a la necesidad que hemos tenido de introducir la matriz de rotación en los cálculos. Este acoplamiento de los modos de translación y rotación será determinante a la hora de diseñar el controlador del vehículo.

7 SISTEMA DE CONTROL IMPLEMENTADO POR EL FABRICANTE

7.1 Planteamiento

Un aspecto importante para llevar a cabo con éxito los pasos posteriores de este trabajo ha consistido en un estudio detallado del sistema de control desarrollado por la compañía diseñadora del Crazyflie 2.0, que se carga en el vehículo con el firmware del microcontrolador principal del mismo. El conocimiento adquirido en esta fase se ha empleado en varias fases posteriores del trabajo, tanto en la implementación de la parte del controlador de la simulación por ordenador, como en la validación completa de dicha simulación. Dicho conocimiento también es necesario para llevar a cabo la implementación final del diseño de nuevos controladores para el vehículo, una vez probados en el simulador, ya que dichos controladores necesitan introducirse finalmente en el firmware del microcontrolador principal y, para ello, se debe entender cómo funciona dicho código y cómo es posible modificarlo para introducir cualquier nuevo controlador.

Para entender el diseño del firmware se ha seguido una técnica de ingeniería inversa, realizando, en primer lugar, un estudio detallado del código fuente, que la compañía fabricante tiene archivado en un repositorio de acceso público en GitHub [30]. Una vez identificados en el código los módulos responsables de la función de control se han extraído de los mismos las variables y ecuaciones principales que conforman el controlador del vehículo.

Tras un análisis detallado se ha verificado que el controlador desarrollado por el fabricante es un controlador lineal basado en diversos controladores PID que asume un modelo de planta linealizado en el entorno de la configuración de vuelo a punto fijo, y que adopta una estructura de control ya clásica para este tipo de vehículos, de tres elementos de control anidados jerárquicamente. En primer lugar está el controlador de posición que proporciona parte de la señal de referencia al controlador de la planta motriz y que al mismo tiempo manda una señal de referencia al controlador de orientación. El controlador de orientación, por su parte, genera el resto de la señal de referencia que el controlador de planta motriz precisa para generar la señal de mando que es la que se transmite a los actuadores. El controlador de posición y el de orientación se engloban conceptualmente en el controlador de localización, que es el que proporciona la señal de referencia deseada al controlador de la planta de potencia. En la Tabla 16 se muestran los bloques conceptuales básicos del sistema de control del vehículo que son los que se han empleado en el apartado 8.4.2. Se trata, por otra parte, de los dos bloques habituales en la amplia mayoría de los robots móviles.

Tabla 16 : Bloques integrantes del sistema de control

Bloque	Entrada	Salida
Controlador de localización	-Posición vehículo (real y deseada) -Orientación vehículo (real y deseada)	-Fuerza aerodinámica de control (deseada) -Momento aerodinámico de control (deseado)
Controlador de planta de potencia	-Fuerza aerodinámica de control (deseada) -Momento aerodinámico de control (deseado)	PWM (4)

7.2 Firmware del microcontrolador STM32F405

El código fuente diseñado por Bitcraze para ser cargado en el microcontrolador principal del vehículo consta de unos 500 archivos escritos en lenguaje C. Entre dichos archivos de incluye el sistema operativo de tiempo real freeRTOS [90], los drivers necesarios para los distintos componentes electrónicos, los archivos de configuración, las bibliotecas necesarias y, finalmente, los módulos que llevan a cabo las distintas tareas encomendadas al microcontrolador, tanto las propias de control, como las de lectura de sensores, estimación de estado, interface con el otro controlador, interface con las placas de expansión, etc.

7.2.1 Versión empleada

El control de versiones de todo el software relacionado con el Crazyflie 2.0 está centralizado en el repositorio GitHub [30]. En lo referente al firmware del microcontrolador STM32F405, el equipo de desarrolladores del fabricante se ha mostrado bastante activo en su mantenimiento desde la salida al mercado del producto, realizando frecuentes cambios en el mismo, algunos de importancia significativa, y liberando unas dos versiones consolidadas al año.

Como consecuencia de lo anterior, un aspecto importante del trabajo consistió en decidir la versión del firmware que se iba a emplear. Tras tomar la decisión se actualizó el firmware del microcontrolador con la versión elegida y se tuvo cuidado de no volver a actualizarlo a lo largo del trabajo y de realizar el estudio del firmware sobre dicha versión. En la Tabla 17 se muestran los detalles de la versión elegida. El archivo de actualización contiene el firmware de los dos microcontroladores del vehículo, pero tal y como se ha mencionado anteriormente, es en el microcontrolador STM32F405 en el que se realizan las funciones relacionadas con el presente trabajo.

Tabla 17 : Versión del firmware del microcontrolador STM32F405 empleada [91]

Versión (release)	2016.09
Fecha	08.09.2016
Localización	https://github.com/bitcraze/crazyflie-firmware
Rama (branch)	master
Etiqueta (tag)	2016.09
Archivo de actualización (nombre)	`crazyflie-2016.11.zip`
Archivo de actualización (contenido)	Crazyflie 1.0/2.0 firmware release 2016.09 Crazyflie 2.0 NRF Firmware 2016.11

Por otro lado, el firmware y su interface con el cliente de la estación de tierra es configurable respecto a varias de las variables de las que dispone, como pueden ser modos de vuelo, modos de estabilización, etc. En el presente trabajo se ha empleado una configuración fija tanto en la simulación como en los ensayos en vuelo. En el apartado 7.2.4 se proporcionan más detalles de la configuración empleada.

7.2.2 Análisis de la estructura

El firmware del controlador en su versión 2016.09 presenta la estructura de directorios mostrada en la Tabla 18.

Tabla 18 : Estructura de directorios del firmware del microcontrolador STM32F405 [91]

```
./              | Root, contains the Makefile
 + init         | Contains the main.c
 + config       | Configuration files
 + drivers      | Hardware driver layer
 |  + src       | Drivers source code
 |  + interface | Drivers header files. Interface to the HAL layer
 + hal          | Hardware abstaction layer
 |  + src       | HAL source code
 |  + interface | HAL header files. Interface with the other parts of the program
 + modules      | Firmware operating code and headers
 |  + src       | Firmware tasks source code and main.c
 |  + interface | Operating headers. Configure the firmware environement
 + utils        | Utils code. Implement utility block like the console.
 |  + src       | Utils source code
 |  + interface | Utils header files. Interface with the other parts of the program
 + platform     | Platform specific files. Not really used yet
 + tools        | Misc. scripts for LD, OpenOCD, make, version control, ...
 |              | *** The two following folders contains the unmodified files ***
 + lib          | Libraries
 |  + FreeRTOS  | Source FreeRTOS folder. Cleaned up from the useless files
 |  + STM32...  | Library folders of the ST STM32 peripheral libs
 |  + CMSIS     | Core abstraction layer
```

Para entender el funcionamiento del firmware se ha tenido que realizar un análisis detallado del mismo, ya que no existía documentación explicativa y la cantidad de comentarios en el código era muy escasa y en varias ocasiones contenía errores tipográficos debidos al copiado y pegado de fragmentos de código sin modificar los comentarios. De forma general, en una primera fase se han identificado los módulos de código más directamente relacionados con el controlador de localización y el de los motores y se ha establecido su jerarquía, basada en niveles. Se ha encontrado que la mayoría de los módulos de código relevantes se encuentran en el directorio `modules` y alguno de ellos en el directorio `drivers`. Por otro lado, el módulo `stabilizer.c` es el módulo principal desde el que se realizan las llamadas a la mayoría de los módulos que implementan las distintas tareas de control del vehículo. Por supuesto, el funcionamiento del firmware comienza con el módulo `main.c` que lanza el sistema operativo, configura la plataforma y la inicializa, y también configura el sistema y lo inicializa incluyendo los relojes e interrupciones. El firmware ejecuta durante su funcionamiento una serie de tareas gestionadas por el sistema operativo con diferentes prioridades. En el archivo `config.h` se asignan las prioridades a los grupos de tareas, teniendo la máxima prioridad la tarea `SYSLINK` que crea los paquetes de datos a partir de la información recibida de la estación de tierra a través del microcontrolador STM32F405. En el siguiente nivel de prioridad se encuentran las tareas `STABILIZER` y `SENSORS` responsables respectivamente de la estabilización del vehículo y de la gestión de los datos medidos por los sensores de a bordo.

7.2.3 Jerarquía de módulos principales

En la Figura 23 se muestra el resultado del análisis realizado del firmware. Se muestran únicamente los módulos que son relevantes para las actividades de control de localización (posición y actitud) del vehículo. Se puede observar que el código se ha diseñado de forma modular, con varios niveles jerárquicos conectados mediante llamadas a funciones. La ejecución

del código se inicia en el nivel 0 con el módulo `main.c` y va evolucionando hacia los niveles inferiores. En el nivel 2 se encuentra el módulo `stabilizer.c` que contiene llamadas a los módulos del nivel 3 que implementan las funciones esenciales de control:

- Lectura de la señal de los sensores: `sensors_stock.c`
- Estimación del estado de la planta: `estimator_complementary.c`
- Lectura de la señal de referencia: `commander.c`
- Generación de la señal de control: `controller_pid.c`
- Generación de la señal de control para cada motor: `power_distribution_stock.c`

De igual forma, los módulos de nivel 3 realizan llamadas a módulos de niveles inferiores como, por ejemplo, al módulo `pid.c` que implementa un controlador PID y que es llamado sistemáticamente tanto por el controlador de posición como por el controlador de orientación.

Todos los módulos antes mencionados incluyen una función de inicialización y otra de test que son lanzadas al arrancar el vehículo.

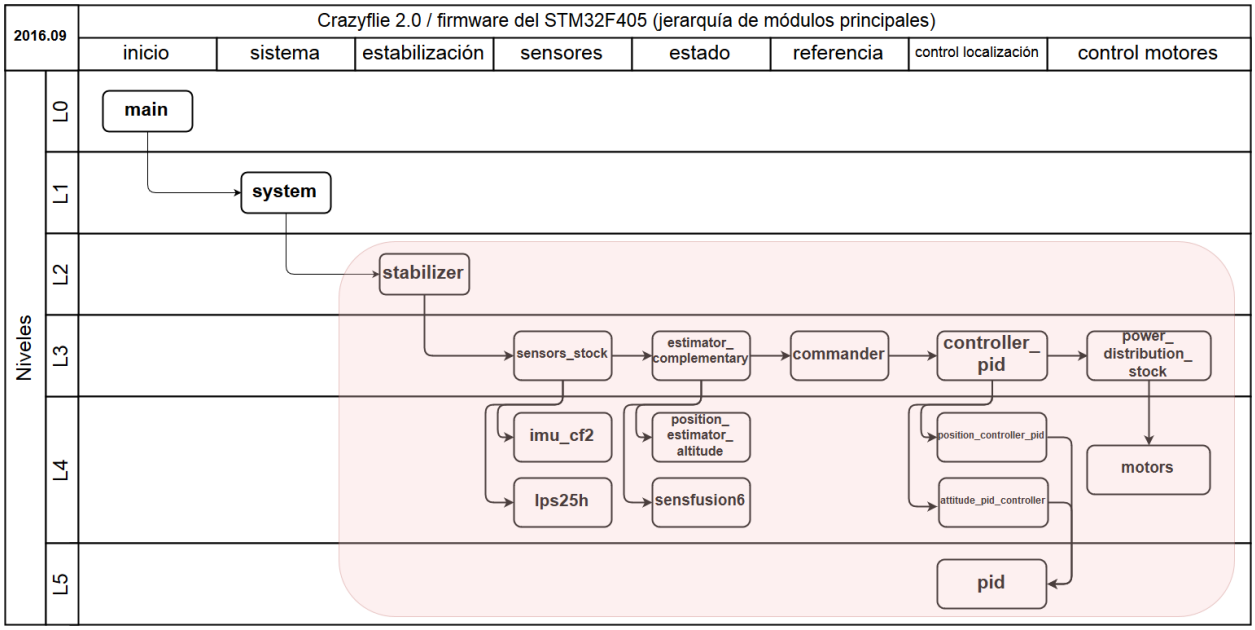


Figura 23 : Módulos del firmware del STM32F504 relacionados con los controladores

7.2.4 Configuración y parámetros

Hay que tener en cuenta que el código del firmware fue originalmente desarrollado para el primer producto que sacó al mercado la compañía diseñadora, denominado Crazyflie, el cual presentaba unas características dinámicas significativamente diferentes a las del Crazyflie 2.0, por ejemplo tenía una configuración en +, en contraposición a la configuración en X del Crazyflie 2.0. Los desarrolladores reutilizaron dicho código posteriormente para que sirviera para ambos productos, el Crazyflie y el Crazyflie 2.0, introduciendo numerosas directivas condicionales en el código que se configuran en el archivo `make` en tiempo de compilación dependiendo del producto en el que pretendamos cargar el firmware. Por otro lado, ya centrándonos en el vehículo objeto de estudio, el firmware permite configurar numerosas opciones para variar las características de vuelo del vehículo, así como todos los parámetros del controlador. En el presente trabajo se ha optado por mantener una configuración y conjunto de

parámetros fijos. En el ANEXO 2 se han proporcionado más detalles de la configuración del firmware empleada, así como los listados de los parámetros y variables del mismo que están relacionados directamente con la función de control. Dicha información puede ser necesaria para comprender correctamente las leyes de control incluidas como ecuaciones en los apartados 7.3 y 7.4, ya que en dichos apartados se han considerado los controladores en la configuración selecciona.

7.3 Controlador de localización

Tal y como se puede observar en la Figura 23, el módulo `controller_pid.c` constituye el controlador de localización del vehículo. El módulo `stabilizer.c` realiza una llamada a la función `stateController` en uno de sus pasos. Dicha función está situada en el módulo `controller_pid.c`, y recibe del nivel superior los valores necesarios de las siguientes variables: referencia, estado y control. A su vez, la función `stateController` realiza llamadas a las funciones contenidas en los niveles inferiores que contienen las leyes de control de posición y orientación descritas en los apartados 7.3.1 y 7.3.2. Además de realizar las mencionadas llamadas, la función mantiene el ángulo de guiñada deseado, o de referencia, dentro del intervalo ±180º según la ley descrita en la ecuación (20).

$$attitudeDesired.yaw = \begin{cases} attitudeDesired.yaw - 360° & attitudeDesired.yaw > 180° \\ attitudeDesired.yaw & -180° \leq attitudeDesired.yaw \leq 180° \\ attitudeDesired.yaw + 360° & attitudeDesired.yaw < -180° \end{cases} \quad (20)$$

donde: $attitudeDesired.yaw$ ángulo de guiñada deseado medido en grados

Finalmente, la función también se encarga de resetear el controlador de orientación cuando la señal de control de empuje es nula, a través de la función `attitudeControllerResetAllPID`, para asegurar que el control de orientación siempre esté vinculado a la existencia de giro en los actuadores, ya que el mencionado control se basa en las diferencias en la velocidad de rotación de los actuadores y no puede llevarse a cabo si los motores están parados.

Una vez que se han realizado todas estas tareas (control de posición, control de orientación y tareas adicionales), el controlador de localización le pasa la señal de fuerza y momento deseado al controlador de la planta de potencia a través de la variable control, tal y como se detalla en la Tabla 19.

Tabla 19 : Variables de interface controlador localización-controlador planta de potencia

Variable	Notación firmware	Notación simulador
Fuerza aerodinámica deseada eje Z_B	control.thrust	U1
Momento aerodinámico deseado eje X_B	control.roll	U2
Momento aerodinámico deseado eje Y_B	control.pitch	U3
Momento aerodinámico deseado eje Z_B	control.yaw	U4

7.3.1 Control de posición

El controlador de posición (`position_controller_pid.c`) se ejecuta cuando el nivel superior realiza la llamada a la función `positionController` que está contenida en dicho módulo. A continuación, la función realiza tres llamadas al módulo inferior que implementa el controlador PID (`pid.c`) para las tres variables de posición (x, y, z), identificando la variable z con el empuje (`thrust`). Las llamadas al controlador PID se realizan a través de la función `runPID`, que por su parte realiza llamadas a las funciones `pidInit`, `pidSetDesired` y `pidUpdate`, todas ellas pertenecientes al módulo del controlador PID.

Los tres controladores PID independientes generan la señal de control a partir de la posición deseada y la posición real. Una vez obtenida dicha señal (`x, y, newThrust`), el controlador de posición realiza algunas operaciones para pasar finalmente al controlador de orientación los ángulos de alabeo y cabeceo deseados (`attitude.roll, attitude.pitch`) y al controlador de la planta de potencia la tracción que se tiene que generar (`thrust`). A continuación se describen dichas operaciones, no considerándose relevante la descripción del funcionamiento del módulo que contiene el controlador PID.

La primera operación consiste realizar un cambio de sistema de referencia de la señal de control obtenida por el controlador de posición para el plano XY, transformándola en la señal de control alabeo-cabeceo. Para ello se parte del valor del ángulo de guiñada estimado (yawRad) y se emplean las ecuaciones (21), obteniéndose los ángulos de alabeo y cabeceo deseados.

$$\begin{aligned} attitude.roll &= -y \cdot \cos(yawRad) + x \cdot sen(yawRad) \\ attitude.pitch &= -x \cdot \cos(yawRad) - y \cdot sen(yawRad) \end{aligned} \tag{21}$$

donde:

	x	variable intermedia generada por el PID de posición x
	y	variable intermedia generada por el PID de posición y
	$yawRad$	ángulo de guiñada estimado
	$attitude.roll$	ángulo de alabeo deseado
	$attitude.pitch$	ángulo de cabeceo deseado

A continuación se implementa la saturación de dichos ángulos deseados, para evitar que los ángulos crezcan por encima del valor límite definido y poder mantener la hipótesis de controlador lineal, que requiere que los ángulos de las magnitudes afectadas por funciones trigonométricas sean pequeños para poder aproximar el valor del coseno a la unidad y el del seno al ángulo y, de esta forma, usar una aproximación lineal en el control del modelo. En la ecuación (22) se muestran las ecuaciones que implementan dicha saturación que evita que el vehículo efectúe maniobras en las que se supere el ángulo límite tanto en alabeo como en cabeceo.

$$\begin{aligned} attitude.roll &= \begin{cases} rpLimit & attitude.roll \geq rpLimit \\ attitude.roll & -rpLimit < attitude.roll < rpLimit \\ -rpLimit & attitude.roll \leq -rpLimit \end{cases} \\ attitude.pitch &= \begin{cases} rpLimit & attitude.pitch \geq rpLimit \\ attitude.pitch & -rpLimit < attitude.pitch < rpLimit \\ -rpLimit & attitude.pitch \leq -rpLimit \end{cases} \end{aligned} \tag{22}$$

donde:

	$rpLimit$	valor máximo del ángulo de alabeo/balanceo

$attitude.roll$	ángulo de alabeo deseado
$attitude.pitch$	ángulo de cabeceo deseado

Finalmente, se calcula el valor de la variable de tracción que se va a enviar al controlador de la planta de potencia partiendo del valor de la señal de control proporcionado por el controlador PID (en base a la diferencia entre el valor de z deseado y el estimado). Dicha señal se va a codificar como un entero de 16 bits sin signo (uint16), correspondiendo el valor de 0 al motor parado y el valor de 65535 al motor a máximo funcionamiento. En primer lugar se le añade a la señal de control el valor de tracción necesario para el vuelo a punto fijo (`THRUST_BASE`), que es el que proporciona una tracción que es idéntica al peso del vehículo. A continuación se satura dicha señal para evitar que crezca por encima de un límite de seguridad establecido por el diseñador (`THRUST_UPPER_LIMIT`). El valor de saturación de la señal (45000) se corresponde con el 68% del valor máximo posible. Cuando los 4 motores del vehículo están a su valor máximo de tracción se genera una fuerza resultante de 0.408 N [4] [92], que compensa los 42 g de peso máximo al despegue del vehículo mencionados en el apartado 4.2.1 y le permite despegar. Si el vehículo pesa más, no es posible el despegue porque el controlador no permite a la planta propulsora conseguir una mayor tracción, a pesar de que aún existe algo de margen en los motores para dar más potencia. Finalmente el valor calculado (`thrust`) es el que se envía al controlador de la planta de potencia. En la ecuación (23) se muestran los cálculos mencionados.

$$thrust = newThrust + THRUST\_BASE$$
$$thrust = \begin{cases} \text{THRUST\_UPPER\_LIMIT} & thrust \geq THRUST\_UPPER\_LIMIT \\ thrust & thrust < \text{THRUST\_UPPER\_LIMIT} \end{cases} \quad (23)$$

donde:

$THRUST\_BASE$	valor de fuerza para vuelo a punto a punto fijo
$THRUST\_UPPER\_LIMIT$	valor máximo de fuerza
$newThrust$	variable intermedia generada por el PID z
$thrust$	fuerza aerodinámica deseada en el eje Z_B

7.3.2 Control de orientación

El controlador de orientación (`attitude_pid_controller.c`), se ejecuta, al igual que el controlador de posición, cuando el nivel superior realiza una llamada a la función que lo implementa. En este caso existen dos funciones y dos grupos de controladores para cada uno de los ejes del sistema $O_BX_BY_BZ_B$: el controlador del ángulo de Euler, a través de la llamada a la función `attitudeControllerCorrectAttitudePID` y el controlador de su variación temporal a través de la llamada a la función `attitudeControllerCorrectRatePID`. Ambas funciones están contenidas en dicho módulo. Las llamadas se producen de forma secuencial: primero se realiza la llamada a la función de control del ángulo de Euler pasándole los valores de los tres ángulos de Euler deseados y a continuación se realiza una llamada a la función de control de su variación temporal. La secuencia se describe a continuación.

A la primera función (`attitudeControllerCorrectAttitudePID`), el nivel superior le pasa, además del valor estimado del estado, los valores deseados de los ángulos de Euler generados por el controlador de posición y el nivel superior (véase apartados 7.3 y 7.3.1). Dicha función realiza tres llamadas al módulo inferior que implementa el controlador PID (`pid.c`) para los tres ángulos de Euler (`Roll, Pitch, Yaw`). Las llamadas al controlador PID se realizan a través de las funciones `pidSetDesired`, `pidUpdate` en el caso de alabeo y

balanceo y las funciones `pidSetError` y `pidUpdate` para la guiñada, todas ellas pertenecientes al módulo del controlador PID. En el caso del eje de guiñada, es la función la que calcula el error en el ángulo de guiñada y no el controlador PID porque la función también realiza una manipulación del valor del error para mantenerlo dentro del intervalo de ±180º de una forma similar a lo descrito en la ecuación (20) del apartado 7.3, antes de lanzar el controlador PID. En la ecuación (24) se muestra dicho ajuste del error de guiñada.

$$yawError = \begin{cases} yawError - 360° & yawError > 180° \\ yawError & -180° \leq yawError \leq 180° \\ yawError + 360° & yawError < -180° \end{cases} \quad (24)$$

donde: $yawError$ señal de error del ángulo de guiñada medida en grados

Finalmente, los tres controladores PID independientes (ángulos de alabeo, balanceo y guiñada) generan la señal que contiene las variaciones temporales de los ángulos de Euler deseadas, que es la que usa la segunda función que se describe a continuación.

La segunda función (`attitudeControllerCorrectRatePID`) recoge los valores de las variaciones deseadas generados por la primera función y las variaciones disponibles en el vector de estado que le pasa el nivel superior. Dicha función realiza tres llamadas al módulo inferior que implementa el controlador PID (`pid.c`) para las tres variaciones temporales de los ángulos de Euler (`rollRate, pitchRate, yawRate`). Las llamadas al controlador PID se realizan a través de las funciones `pidSetDesired` y `pidUpdate`. Una vez obtenidos los valores correspondientes de los tres controladores PID se implementa la saturación de la señal de salida para mantenerla dentro del intervalo de ±INT16 MAX. Para ello se realizan los cálculos mostrados en la ecuación (25).

$$rollOutput = \begin{cases} INT16\_MAX & rollOutput \geq INT16\_MAX \\ rollOutput & -INT16\_MAX < rollOutput < INT16\_MAX \\ -INT16\_MAX & rollOutput \leq -INT16\_MAXt \end{cases}$$

$$pitchOutput = \begin{cases} INT16\_MAX & pitchOutput \geq INT16\_MAX \\ pitchOutput & -INT16\_MAX < pitchOutput < INT16\_MAX \\ -INT16\_MAX & pitchOutput \leq -INT16\_MAXt \end{cases} \quad (25)$$

$$yawOutput = \begin{cases} INT16\_MAX & yawOutput \geq INT16\_MAX \\ yawOutput & -INT16\_MAX < yawOutput < INT16\_MAX \\ -INT16\_MAX & rollOutput \leq -INT16\_MAXt \end{cases}$$

donde:

$INT16\_MAX$	valor máximo de un entero de 16 bits con signo
$rollOutput$	momento aerodinámico deseado eje X_B
$pitchOutput$	momento aerodinámico deseado eje Y_B
$yawOutput$	momento aerodinámico deseado eje Z_B

Finalmente, la señal de salida del controlador de orientación (`rollOutput, pitchOutput, yawOutput`) es la que se va a enviar al controlador de la planta de potencia y es la que contiene los momentos deseados en cada uno de los ejes del sistema de referencia que tendrá que generar el otro controlador. La interface entre el controlador de localización y el controlador de la planta de potencia se ha detallado en la Tabla 19.

7.4 Controlador de la planta de potencia

El controlador de la planta de potencia diseñado por el fabricante está compuesto de dos bloques conceptuales diferenciados: el distribuidor y el controlador.

El bloque distribuidor está codificado en el archivo `power_distribution_stock.c` y se encarga de asignar la señal de mando a cada uno de los motores a partir de la señal de salida del controlador de localización, es decir, a partir de la variable `control`, que es la que contiene la fuerza y par deseados. El archivo `stabilizer.c` llama a la función `powerDistribution` tras haber actualizado el valor de la variable `control` tal y como se ha descrito en el apartado 7.3, y es dicha función la que realiza la distribución de la señal de mando a los cuatro actuadores del sistema empleando las relaciones descritas en la ecuación (26). Ambas relaciones aseguran que los actuadores generarán la fuerza y momento demandados y que no se supera la señal de mando máxima.

$$\begin{pmatrix} motorPower.m1 \\ motorPower.m2 \\ motorPower.m3 \\ motorPower.m4 \end{pmatrix} = \begin{pmatrix} 1 & -\frac{1}{2} & \frac{1}{2} & 1 \\ 1 & -\frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & \frac{1}{2} & -\frac{1}{2} & 1 \\ 1 & \frac{1}{2} & \frac{1}{2} & -1 \end{pmatrix} \begin{pmatrix} control.thrust \\ control.roll \\ control.pitch \\ control.yaw \end{pmatrix} \tag{26}$$

$$motorPower.m = \begin{cases} limitUint16 & motorPower.m \geq limitUint16 \\ motorPower.m & motorPower.m < limitUint16 \end{cases}$$

donde:

$limitUint16$	valor máximo de un entero de 16 bits sin signo
$control.thrust$	fuerza aerodinámica deseada eje Z_B
$control.roll$	momento aerodinámico deseado eje X_B
$control.pitch$	momento aerodinámico deseado eje Y_B
$control.yaw$	momento aerodinámico deseado eje Z_B
$motorPower.m1$	señal de control del motor 1
$motorPower.m2$	señal de control del motor 2
$motorPower.m3$	señal de control del motor 3
$motorPower.m4$	señal de control del motor 4

La distribución realizada de acuerdo a la ecuación (26) es consistente con la configuración en X del Crazyflie 2.0, los sentidos de giro de los motores y los criterios de signos establecidos para los momentos, tal y como se mostró en la Figura 18. De hecho la ecuación tiene la misma estructura de las matrices de asignación del apartado 6.8 pero invertida y con los signos de cabeceo y guiñada cambiados, tal y como se ha explicado en el apartado 7.3.

El bloque controlador está codificado en uno de los archivos de drivers del sistema, concretamente en el módulo `motors.c`. Su cometido es generar la señal de voltaje PWM que se suministra a cada uno de los motores del vehículo. El archivo

`power_distribution_stock.c` llama a la función correspondiente (`motorsSetRatio`) a través de la misma función que implementa el bloque distribuidor (`powerDistribution`), una vez se han calculado los valores de para cada uno de los motores en el bloque distribuidor y se han guardado en la variable `motorPower`. En las ecuaciones (27) y (28) se reproduce la ley de control de los motores que tiene en cuenta tanto el voltaje de suministro de la batería, como la conversión a porcentaje y la dependencia lineal y cuadrática que presenta las fuerzas y momentos con la velocidad de giro de los motores. Tal y como se ha comentado anteriormente, estas ecuaciones, junto con la ecuación (26) del distribuidor constituyen la función inversa a la descrita en el apartado 6.8.

$$\begin{pmatrix} thrust_1 \\ thrust_2 \\ thrust_3 \\ thrust_4 \end{pmatrix} = \begin{pmatrix} motorPower.m1 \\ motorPower.m2 \\ motorPower.m3 \\ motorPower.m4 \end{pmatrix} \cdot \frac{60}{limitUint16} \tag{27}$$

$$\begin{pmatrix} PWM_1 \\ PWM_2 \\ PWM_3 \\ PWM_4 \end{pmatrix} = \frac{1}{V_s} \left(Ca \begin{pmatrix} thrust_1 \\ thrust_2 \\ thrust_3 \\ thrust_4 \end{pmatrix}^2 + Cb \begin{pmatrix} thrust_1 \\ thrust_2 \\ thrust_3 \\ thrust_4 \end{pmatrix} \right) \tag{28}$$

donde:

$limitUint16$	valor máximo de un entero de 16 bits sin signo
$motorPower.m1$	señal de control del motor 1
$motorPower.m2$	señal de control del motor 2
$motorPower.m3$	señal de control del motor 3
$motorPower.m4$	señal de control del motor 4
$thrust_i$	variable intermedia de control de cada motor
Ca	constante de conversión de señal de mando a voltaje
Cb	constante de conversión de señal de mando a voltaje
V_s	voltaje proporcionado por la batería
PWM_i	porcentaje de PWM que se envía a cada motor

8 SIMULACIÓN POR ORDENADOR

8.1 Planteamiento

Tras establecer en apartados anteriores las ecuaciones que gobiernan la dinámica del vehículo y sus actuadores, e indagar en las características de los controladores de localización y planta de potencia desarrollados por el fabricante y grabados en la memoria del microcontrolador principal del vehículo en forma de firmware, se está en condiciones de proceder a realizar una simulación por ordenador de la dinámica del vehículo, ya que tras las actividades anteriores se cuenta con el conocimiento necesario de la planta y del controlador como para enfrentar dicha simulación. La herramienta seleccionada para realizarla ha sido el paquete de software Simulink® [36], un entorno de diagramas de bloque para la simulación multidominio y el diseño basado en modelos. Gracias a la posibilidad que tiene Simulink® de integrarse con el paquete de software MATLAB® [35], se ha empleado esta última herramienta para las labores de configuración, preprocesado y postprocesado, así como para la incorporación de ciertos algoritmos en la simulación a través del bloque "función de MATLAB® definida por el usuario", que permite definir funciones que son compiladas y usadas como código embebido por Simulink®. Por otro lado, la simulación del modelo se ha llevado a cabo en su totalidad en Simulink® mediante la resolución numérica de las ecuaciones diferenciales que lo gobiernan. Simulink® cuenta para ello con los solucionadores (solvers) de ecuaciones diferenciales por métodos numéricos disponibles en MATLAB®, cuyos parámetros pueden ser ajustados según las características de las ecuaciones diferenciales que se pretenden resolver.

8.2 Estructura de archivos del simulador

El simulador se ha diseñado mediante una serie de archivos de MATLAB® que realizan las siguientes tareas de forma secuencial:

- Carga de la configuración del simulador y de los parámetros de entorno, planta y controlador
- Preprocesado de los datos de entrada
- Simulación
- Postprocesado de los resultados

El archivo `CrazySimulator_v0.m` es el archivo principal del simulador y es, asimismo, el que realiza las llamadas al preprocesador (`prepro.m`), a la simulación (`Modelo_v0.slx`) y al posprocesador (`postpro.m`).

La carga de la configuración y de los parámetros la realiza el preprocesador mediante la llamada a las siguientes funciones:

- Configuración (`configuracion__simulador.m`)
- Parámetros del entorno (`parametros_entorno.m`)
- Parámetros de la planta (`parametros_planta.m`)
- Parámetros del controlador (`parametros_controlador.m`)
- Parámetros del controlador nuevo (`parametros_controlador_nuevo.m`)

Para usar el simulador se debe establecer primero la configuración del mismo y del entorno de vuelo a emplear en las correspondientes funciones. Asimismo, se precisan fijar también en las funciones correspondientes los parámetros del controlador que vamos a simular, bien sean los pertenecientes al firmware desarrollado por el fabricante o bien a al desarrollado por el usuario como controlador nuevo. Los parámetros de la planta son, en principio, fijos y no hay que cambiarlos a menos que se hayan realizado modificaciones en el vehículo. Si ese fuera el caso, habría que realizar cambios en la función correspondiente. Finalmente, una vez establecida la configuración y los parámetros se debe ejecutar el archivo principal (`CrazySimulator_v0.m`) para lanzar el simulador.

En la Figura 24 se muestra la estructura de archivos anteriormente descrita para el caso en el que usemos la simulación del firmware del fabricante. Si usamos un nuevo controlador baste remplazar el archivo `parametros_controlador.m` por el archivo `parametros_controlador_nuevo.m` y realizar la selección correspondiente en el archivo de configuración.

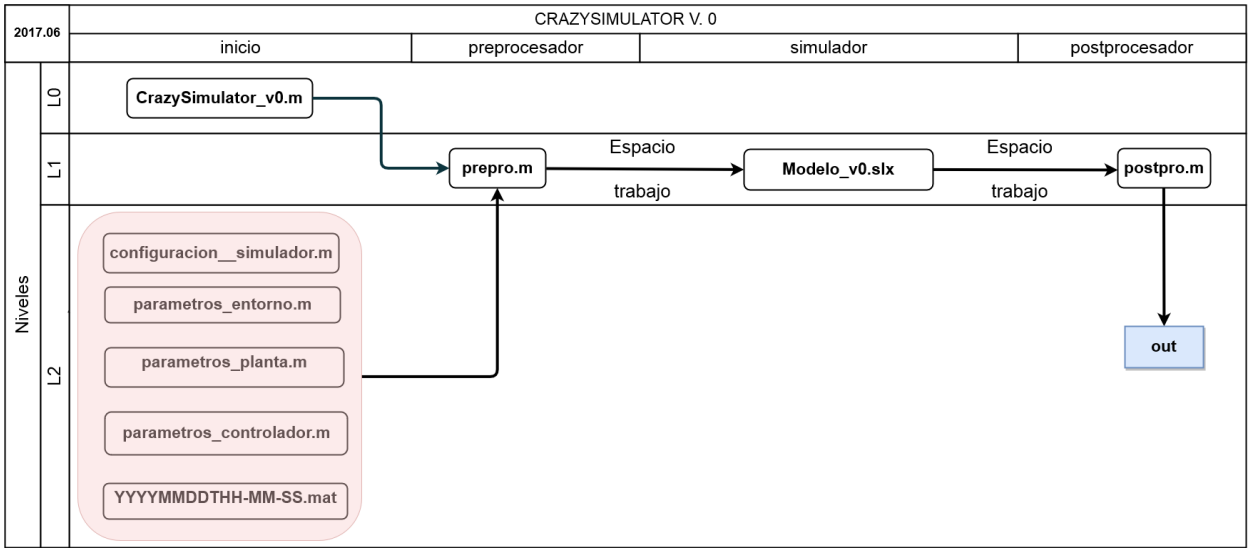


Figura 24 : Estructura archivos simulador (caso firmware fabricante)

En el ANEXO 3 se han reproducido los archivos de MATLAB® que constituyen el simulador. Los archivos incluyen en la cabecera instrucciones para su uso, así como texto de ayuda en el caso de las funciones.

8.3 Preprocesado

La fase de preprocesado se realiza mediante un programa de MATLAB® (`prepro.m`), que lleva a cabo las siguientes tareas:

- Carga de la configuración del simulador (`configuracion__simulador.m`): se realiza una llamada a la función de configuración que fija los valores de las variables de simulación que son necesarios tanto para el resto del preprocesado como para que se pueda ejecutar la simulación posteriormente: selección del modo del simulador, nombre del archivo de datos de telemetría a emplear, selección del tipo de controlador, existencia de entorno, duración de la simulación, señales de referencia, condiciones iniciales de los parámetros, modo de despegue del suelo. Los valores de

configuración quedan cargados en el espacio de trabajo en forma de estructura, que queda así disponible para los módulos de preprocesado y posprocesado, así como para la simulación.

- Carga de los parámetros del entorno (`parametros_entorno.m`): se llama a la función correspondiente en la que se han definido de una forma muy básica las barreras físicas del entorno en el que se va a simular el movimiento del vehículo. El entorno consiste en un tetraedro en el que se definen las posiciones de las caras respecto al sistema de referencia inercial E. A pesar de que en la configuración del simulador podemos optar por eliminar el entorno, su existencia es importante para simular los choques del vehículo con el mismo, y muy especialmente para ser capaces de simular el despegue del vehículo desde su posición estática sobre el suelo, ya que para ello debemos tener dicha cara del tetraedro de entorno correctamente definida. El entorno queda cargado en una estructura en el espacio de trabajo que está disponible para el uso por parte del resto de módulos.
- Carga de los parámetros del modelo de la planta descritos en el apartado 6.5 (`parametros_planta.m`): se lleva a cabo mediante la llamada a la función correspondiente, que carga los parámetros en una estructura que está disponible en el espacio de trabajo para acceso por parte de la simulación y del resto de módulos.
- Carga de los parámetros del controlador elegido, bien mediante la llamada a la función `parametros_controlador.m` en el caso del controlador del fabricante, o bien a través de la función `parametros_controlador_nuevo.m` en el caso de un nuevo diseño. Los parámetros se cargan, también en este caso en una estructura que está disponible en el espacio de trabajo para acceso por parte de los bloques de la simulación, de las funciones embebidas en la misma y del resto de módulos del simulador.
- Carga de datos de vuelo obtenidos por telemetría: se realiza la carga del archivo de datos de vuelo previamente definido en la configuración. Asimismo, en el modo de validación se presenta una figura de los principales parámetros de vuelo y se pide introducir por el teclado la ventana temporal a validar. Una vez se ha seleccionado dicha ventana se llama a la función recorte.m que elimina los datos de vuelo que no están dentro de la ventana y mantiene los que están dentro.
- Preprocesado de datos: se preparan los datos cargados para estar disponibles para su uso por parte de la simulación, ajustando los parámetros de acuerdo a la configuración definida y estableciendo las condiciones iniciales y la duración de la simulación.

El preprocesador se ha reproducido en el ANEXO 3.

8.4 Desarrollo del modelo de simulación

Para proporcionar una mayor claridad respecto a las distintas partes que constituyen el modelo de simulación y la función que realizan, se han establecido cinco niveles de abstracción en el mismo, mediante la agrupación en subsistemas de bloques y señales que realizan funciones parecidas. Asimismo, se han seguido las buenas prácticas establecidas por el desarrollador del software (Mathworks) para mejorar la legibilidad del modelo, como por ejemplo evitar el cruce de líneas de señales. La Figura 25 muestra el nivel de abstracción superior de la simulación (nivel 0), en el que podemos observar los tres bloques principales de la misma que simulan los aspectos principales: la planta, el sistema de control y la estación de tierra. En la simulación planteada, el sistema de control enlaza con la planta a través de los voltajes con PWM enviados a cada uno de los motores, mientras que la planta enlaza con el sistema de control a través del vector de estado directamente, sin incluir sensores. Se ha dejado para futuros desarrollos la simulación de los sensores y la estimación del estado a partir de la fusión de las señales proporcionadas por los mismos. Por otro lado, el bloque que simula la estación de tierra es simplemente un bloque auxiliar de enlace que proporciona a la planta la señal de referencia, o localización deseada, que se envía desde la estación de tierra a través de la Crazyradio PA a la antena del vehículo y que es recogida por el controlador de comunicaciones que, a su vez, la pasa al microcontrolador principal. Este proceso tampoco se ha simulado por no ser relevante para la dinámica, proporcionándose directamente los valores, pero en futuros desarrollos también se podría simular la transmisión de la señal de referencia.

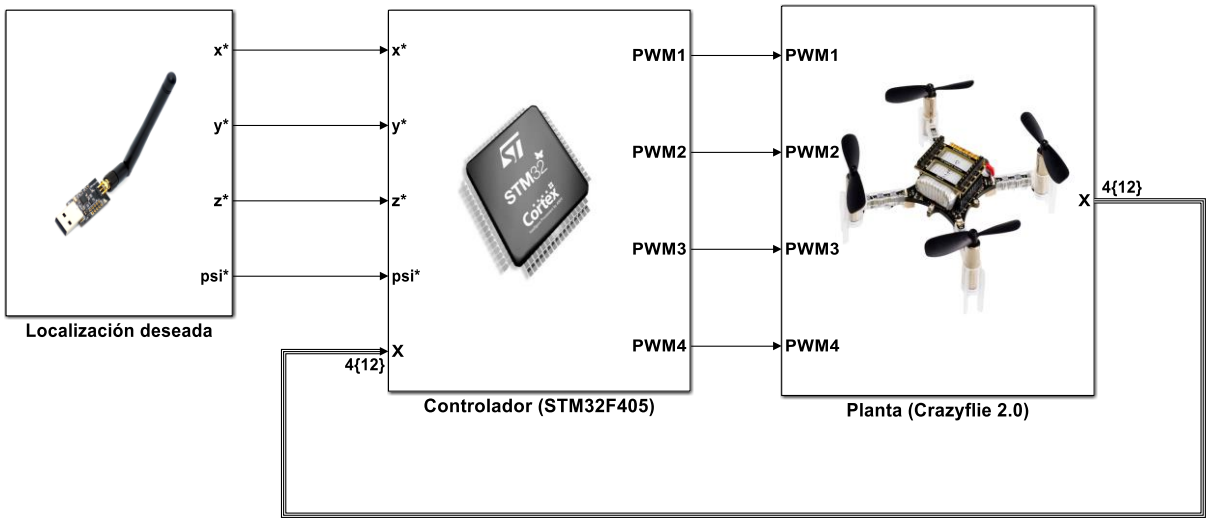


Figura 25 : Modelo simulación (nivel 0)

En la Figura 26 se muestran los distintos componentes del modelo de simulación y se proporciona la visualización necesaria de los niveles de abstracción empleados en su diseño.

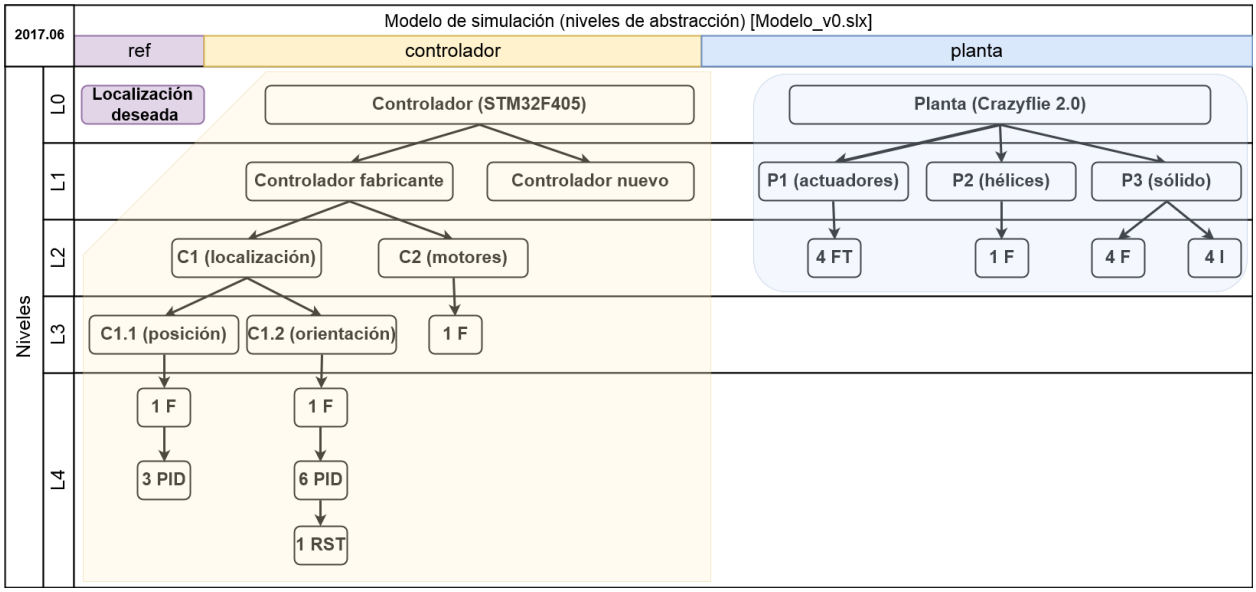


Figura 26 : Niveles de abstracción modelo simulación

F : función MATLAB® embebida **PID**: controlador PID **RST**: función reset
FT : función de transferencia **I** : integrador numérico

8.4.1 Simulación de la planta

En la Figura 27 se muestra el primer subnivel de abstracción (nivel 1) para el caso de la planta. Podemos observar que se han agrupado los elementos en 3 bloques diferenciados. El primero de ellos (P1: dinámica de motores), simula la dinámica de los actuadores en base a la función de transferencia de primer orden establecida en la ecuación (8) del apartado 6.7. Dicho bloque proporciona como salida las velocidades de rotación de los cuatro actuadores que constituyen la entrada del siguiente bloque. El segundo bloque (P2: dinámica de propulsores) implementa la asignación de fuerzas y momentos aerodinámicos descrita en la ecuación (13) del apartado 6.8 mediante una llamada a una función de MATLAB®. Este bloque proporciona como salida las fuerzas y momentos aerodinámicos a los que está sometido el vehículo. Finalmente, el tercer bloque (P3: dinámica aeronave), constituye el elemento esencial de la simulación. En él se implementa el modelo mecánico desarrollado en el apartado 6.9 y su integración numérica, proporcionando como salida el vector de estado de la planta.

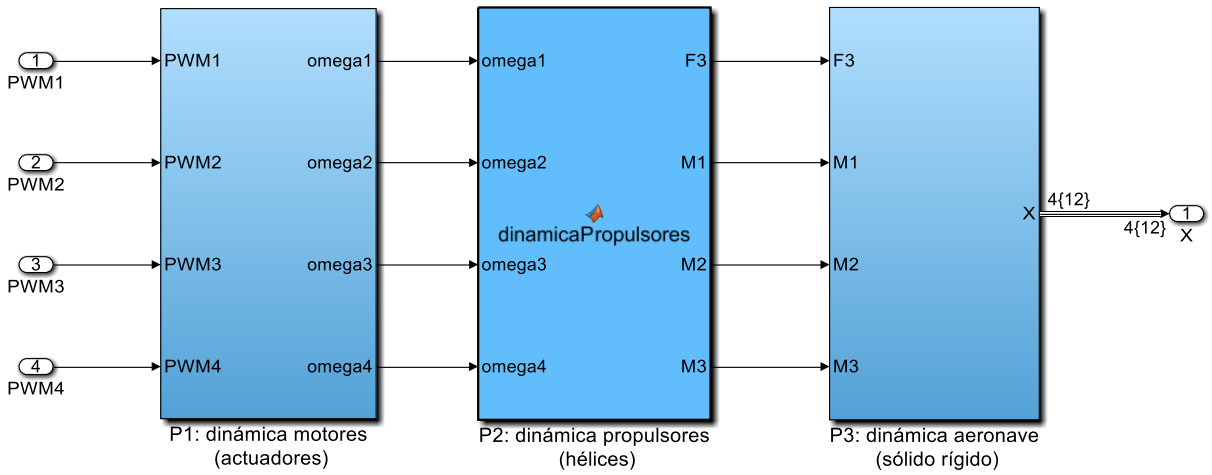


Figura 27 : Modelo de simulación (nivel 1 / Planta)

En la Figura 28 se muestra el segundo subnivel de abstracción (nivel 2) para el caso del bloque que simula el modelo mecánico de la aeronave. Se puede observar la presencia de los cuatro integradores que constituyen el elemento principal de la simulación (en amarillo). Dichos integradores realizan la integración numérica de las ecuaciones diferenciales. En el bloque que implementa el modelado dinámico del sólido rígido se sintetizan, mediante una función de Matlab® las ecuaciones (18) y (19) del apartado 6.9 , mientras que en el bloque de la matriz de transformación se generan las variaciones temporales de los ángulos de Euler a partir de la velocidad angular del vehículo y su orientación, implementando la ecuación (6) del apartado 6.6.

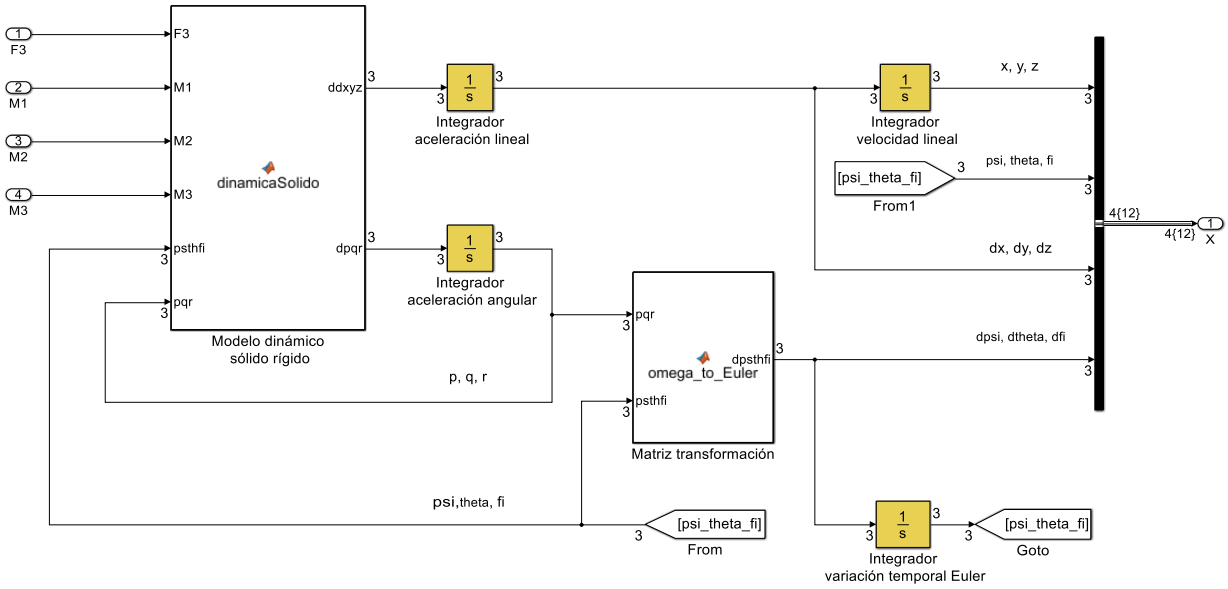


Figura 28 : Modelo de simulación (nivel 2 / Planta / P3: dinámica aeronave)

Finalmente, en la Figura 29 se muestran las cuatro funciones de transferencia que simulan la dinámica de los actuadores y que se sitúan, también en el segundo subnivel de abstracción (nivel 2). Se han establecido de acuerdo al modelo descrito en el apartado 6.7.

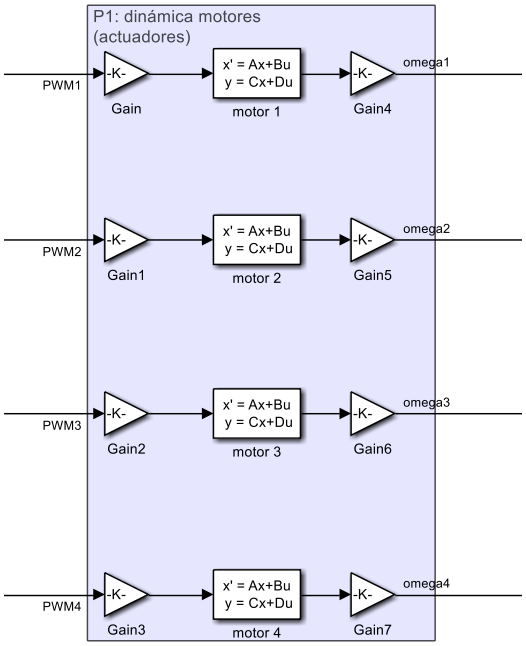


Figura 29 : Modelo de simulación (nivel 2 / Planta / P1: dinámica motores)

8.4.2 Simulación del sistema de control

En el primer subnivel de abstracción del controlador (nivel 1) se ha incluido la posibilidad de emplear o bien el controlador diseñado por el fabricante y codificado en le firmware analizado en el apartado 7, o bien emplear un controlador nuevo en la simulación. En la Figura 30 se muestran los bloques de este subnivel y se puede observar que hay un subsistema que contiene la simulación del firmware analizado y otro subsistema que está disponible para diseñar el nuevo controlador. Un selector comandado por uno de los valores establecidos en el archivo de configuración del simulador conecta el controlador que hemos seleccionado al resto de la simulación.

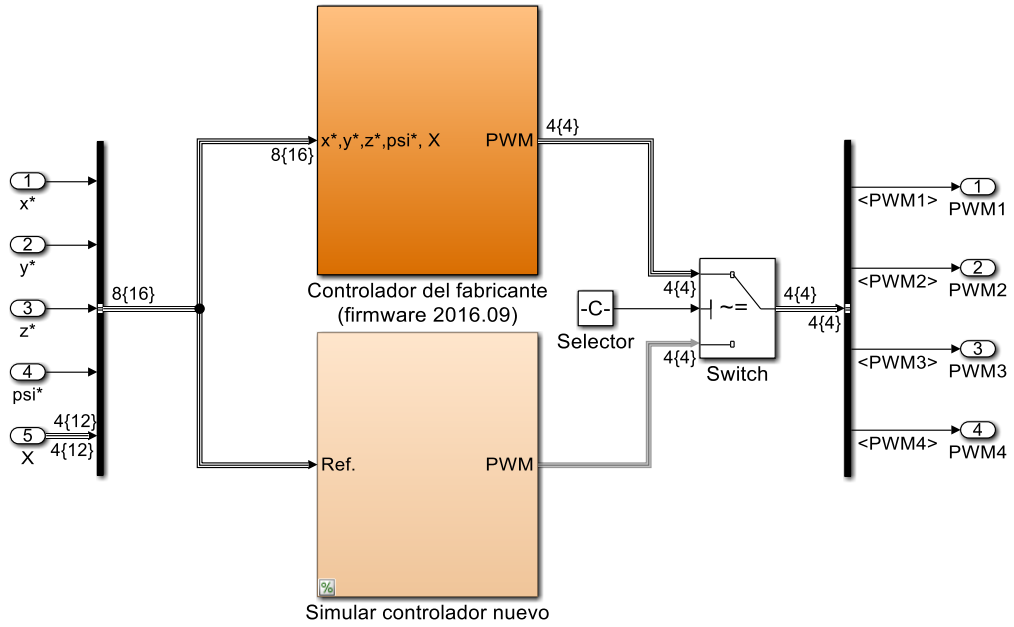


Figura 30 : Modelo de simulación (nivel 1 / Controlador)

En la Figura 31se muestra el segundo subnivel de abstracción (nivel 2) del controlador para el caso del controlador desarrollado por el fabricante. Incluye dos bloques diferenciados. El primero de ellos (C1: controlador de localización del vehículo), recibe la señal de referencia y el vector de estado y, a partir de dichos valores, genera las fuerzas y momentos deseados, tal y como se ha visto en el apartado 7.3. El segundo bloque (C2: controlador motores), implementa mediante una función de Matlab® las ecuaciones (26) (27) y (28) del apartado 7.4. Se trata del bloque que simula el controlador de la planta de potencia que, a partir de los valores deseados para fuerzas y momentos aerodinámicos genera los voltajes de mando de los cuatro actuadores.

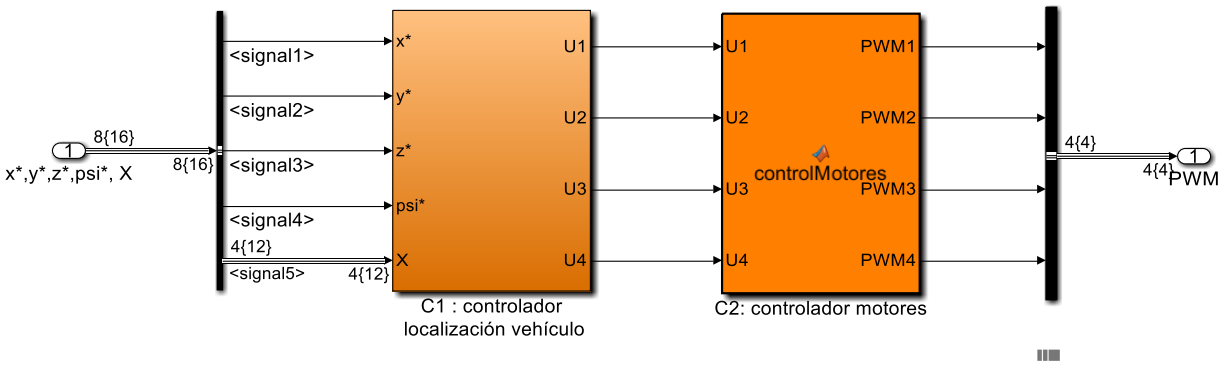


Figura 31 : Modelo de simulación (nivel 2 / Controlador / Controlador del fabricante)

El siguiente subnivel de abstracción (nivel 3) se muestra en la Figura 32 para el caso del bloque que simula el controlador de localización. Se pueden observar los dos controladores (C1.1: posición y C1.2: orientación), que están anidados jerárquicamente tal y como se ha detallado en el apartado 7.1. El vector de estado alimenta a ambos controladores, pero el valor de referencia de orientación se lo proporciona el controlador de posición en la salida al controlador de orientación. Finalmente se puede observar que el controlador de posición proporciona directamente el valor de referencia de fuerza aerodinámica al controlador de los motores, mientras que el valor de referencia de los momentos aerodinámicos lo proporciona el controlador de orientación.

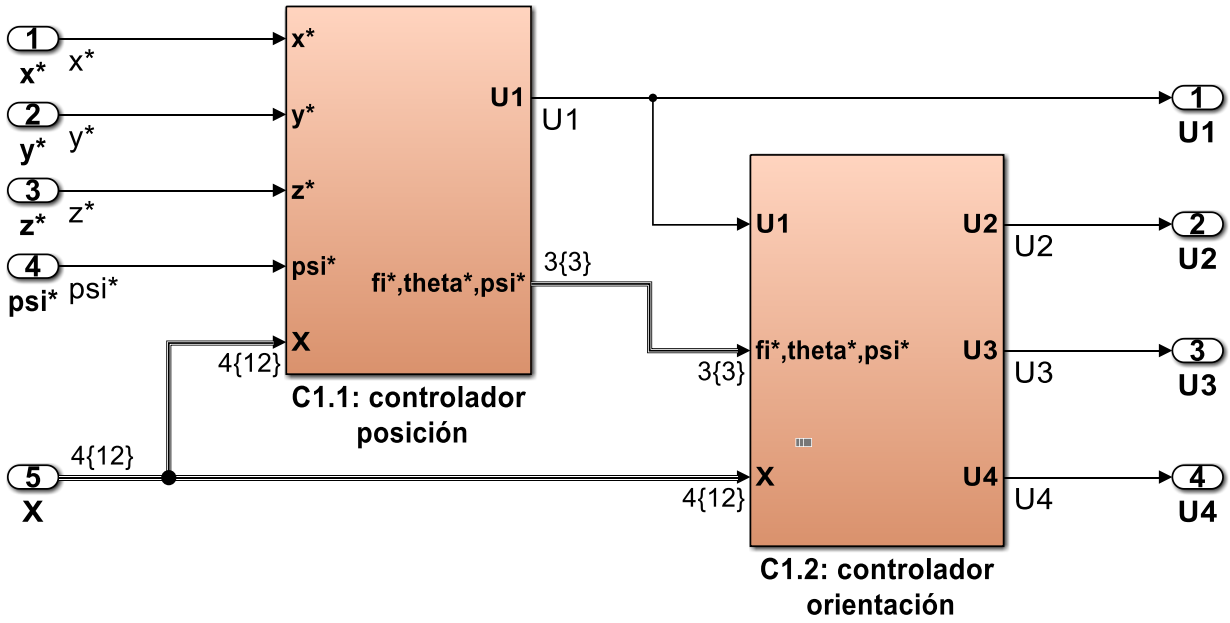


Figura 32 : Modelo de simulación (nivel 3 / Controlador/ C1: controlador localización)

Finalmente el último subnivel de abstracción del modelo de simulación (nivel 4) se muestra en la Figura 33 para el caso del controlador de posición, mientras que el controlador de orientación se muestra en la Figura 34. Ambos bloques simulan las leyes de control detalladas en los apartados 7.3.1 y 7.3.2.

En la Figura 33 podemos observar los 3 controladores PID, uno para cada coordenada que son alimentados tanto por la estación de tierra (referencia) como por el vector de estado (valor real). Asimismo, el subnivel presenta un bloque que contiene la función que realiza un cambio de sistema de referencia de la señal de control del plano horizontal. A su salida podemos visualizar los dos saturadores de los ángulos de alabeo y balanceo deseados, tal y como se ha descrito en el apartado 7.3.1. Finalmente, también podemos observar el bloque encargado de introducir la tracción necesaria para vuelo a punto fijo a partir del momento de despegue tal y como se detalló al final del mismo apartado.

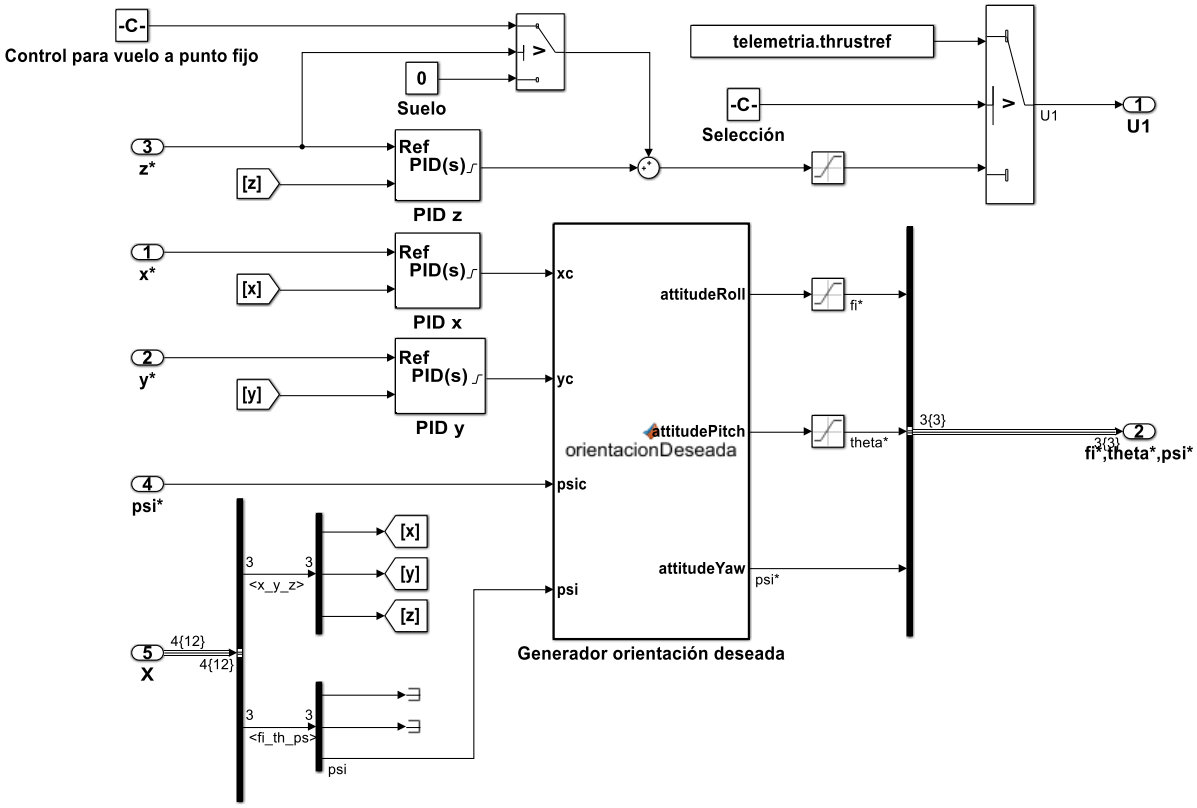


Figura 33 : Modelo de simulación (nivel 4 / Controlador/ C1.1: controlador posición)

En la Figura 33 podemos observar los 6 controladores PID contenidos en el controlador de orientación (uno se encuentra dentro del subsistema de guiñada), que se corresponden con el control del valor de los ángulos de Euler y sus derivadas temporales, el bloque que corrige el error de guiñada, la saturación de las tres señales de salida y el reseteado del controlador de orientación cuando no existe tracción. Todos estos elementos se estudiaron al estudiar el sistema de control implementado por el fabricante en el apartado 7.3.2, y en esta fase se han simulado.

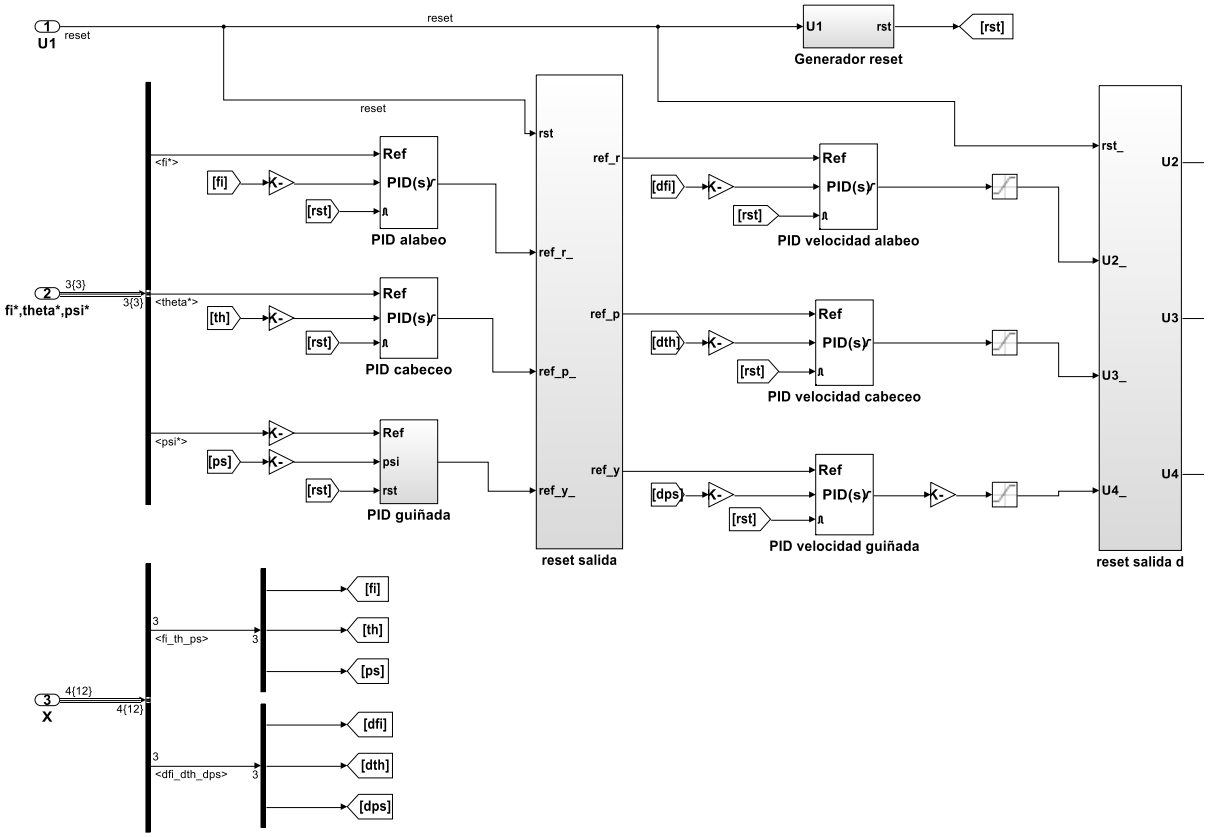


Figura 34 : Modelo de simulación (nivel 4 / Controlador/ C1.2: controlador orientación)

8.4.3 Estación de tierra

La estación de tierra es la encargada de proporcionar al controlador la localización deseada. En el presente trabajo se ha optado por no realizar una modelización completa de la estación y se ha adoptado un esquema simplificado en el que se proporcionan los valores bien a través del archivo de configuración del simulador, en forma de señal en escalón, o bien a través de datos de telemetría. La estación transmite dichos valores al controlador. En la Figura 35 se muestra el sencillo esquema implementado.

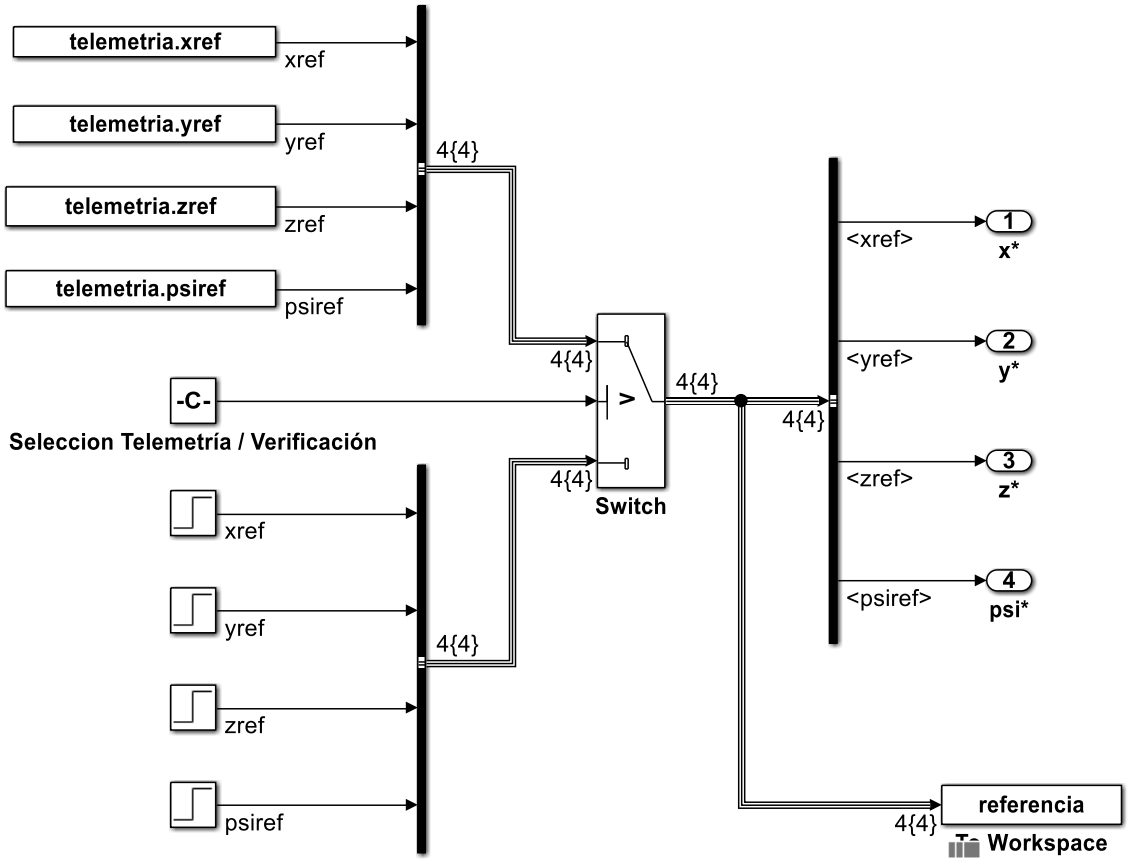


Figura 35 : Modelo de simulación (estación de tierra)

8.4.4 Elementos de interface con el espacio de trabajo de MATLAB®

El modelo de simulación descrito en los apartados anteriores hace uso de la interface nativa entre Simulink® y el espacio de trabajo de MATLAB® que integra ambas herramientas. Dicha interface se aprovecha para cargar en el espacio de trabajo de MATLAB® los parámetros necesarios para el funcionamiento de la simulación en forma de estructuras, tal y como se ha descrito en el apartado 8.3. La simulación al ejecutarse emplea dichos parámetros.

Asimismo, el modelo de simulación se ha completado con bloques adicionales de interface que son necesarios para el intercambio de datos con MATLAB® para el postprocesado y la validación. Para ello se han empleado los bloques “From Workspace” y “To Workspace” de Simulink® que permiten dicho intercambio complementados en algunos casos con interruptores (“Switch”), comandados por los valores establecidos en el archivo de configuración del simulador, que seleccionan la señal a cargar desde el espacio de trabajo en Simulink®. En la Figura 24 aparecen mencionadas estas interfaces preprocesador-modelo y modelo-postprocesador con la expresión “Espacio trabajo”. Asimismo, en la Figura 36 se muestra uno de estos casos. Se trata del vector de estado generado a la salida del modelo de planta. Este vector de estado alimenta al controlador, pero, además se conecta al bloque “To Workspace” mostrado para disponer de su valor a lo largo del tiempo para el postprocesado de la simulaciñon. Más detalles acerca del modelo modificado con los bloques mencionados se pueden encontrar en los apartados correspondientes a postprocesado (8.5) y validación (9.3).

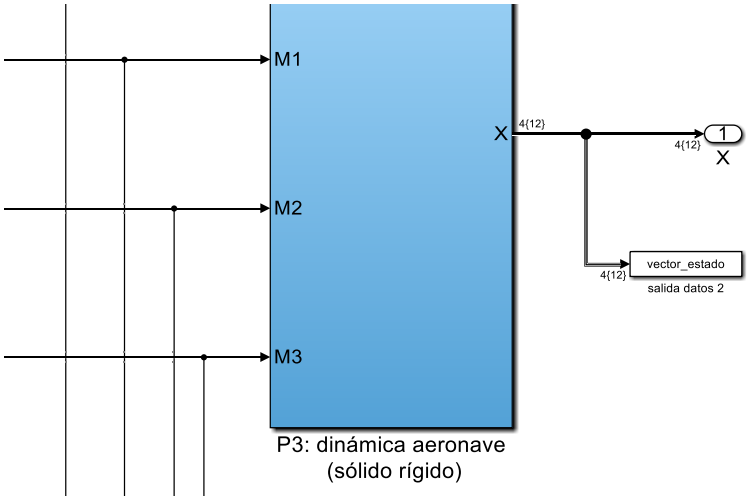


Figura 36 : Detalle elemento de volcado en espacio de trabajo

8.5 Postprocesado

El postprocesado de los datos generados por la simulación se realiza mediante un programa de MATLAB® (`postpro.m`), que prepara los valores y los presenta gráficamente. Para ello cuenta con las siguientes estructuras de datos generados por la simulación y el preprocesador y que se han volcado en el espacio de trabajo de MATLAB®:

- `vector_estado`: se trata de los valores del vector de estado de la aeronave generado por la simulación para cada instante de tiempo.
- `planta_propulsora`: se trata del siguiente conjunto de valores correspondientes a la planta propulsora (motores y hélices) que son generados por la simulación y que se guardan para cada instante de tiempo:
  - Valor de PWM de los cuatro motores (PWM1, PWM2, PWM3, PWM4)
  - Velocidad de rotación del eje de los cuatro motores (omega1, omega2, omega3, omega4)
  - Fuerza y momento aerodinámico resultante (F3, M1, M2, M3)
- `telemetria`: esta estructura de datos contiene los valores de los cuarenta y un parámetros de vuelo capturados por telemetría para cada instante de tiempo. Se genera a partir del archivo correspondiente al vuelo de prueba realizado e identificado en el archivo de configuración. Para más detalles de esta estructura referirse al apartado 9.3.3.
- miscelánea: se emplean en el postprocesado otras estructuras de datos como la estructura `referencia`, que incluye información sobre la señal de referencia de localización deseada, o la estructura `epsilon` que incluye los límites de la eliminación de valores pequeños, así como la función `filtro.m` que realiza el filtrado de datos.

El postprocesador genera las gráficas necesarias a partir de las mencionadas estructuras de datos y discrimina respecto al tipo de actividad que se está realizando (simulación o validación) para presentar las gráficas correspondientes.

El posprocesador es llamado por el programa principal del simulador, tras ejecutar el preprocesador y la simulación. El postprocesador se ha reproducido en el ANEXO 3.

8.6 Visualización

La visualización del simulador es por el momento muy básica. Únicamente se presentan gráficas con los valores de los parámetros tal y como se muestra en los apartados 9.2.3 y 9.3.6. Para mejorar este aspecto, en futuros desarrollos se podría intentar una visualización tridimensional de la simulación en MATLAB®, o incluso conectar el simulador con alguno de los simuladores existentes como Gazebo [93] o V-rep [94].

8.7 Modificaciones en el modelo

A lo largo de las fases de diseño, verificación y validación, el modelo de simulación ha evolucionado notablemente. A pesar de tratarse de un primer prototipo muy básico, ha sido necesario realizar cambios e introducir mejoras al descubrir que la dinámica generada no era consistente. La evaluación de los casos de prueba realizada inicialmente mostró que había algunos elementos que no se habían incluido en el simulador, o que a pesar de estar incluidos contenían errores, cuyo resultado consistía en la generación de resultados incongruentes. Se ha empleado una documento para controlar la configuración del simulador en la fase de diseño, ya que se han producido muchos cambios y evoluciones del mismo. En la presente memoria sólo se presenta la versión final del controlador (ANEXO 3).

Para ilustrar alguna de las modificaciones mayores introducidas, podemos mencionar el bloque que simula el entorno para mantener un registro de los choques, el bloque que simula el suelo, para evitar que el vehículo caiga hacia abajo cuando está en reposo sobre el suelo antes de despegar, o los varios bloques que resetean los controladores PID presentes en el controlador de orientación cuando el vehículo se encuentra en el suelo y aún no ha despegado. Se trata, en definitiva de bloques que tratan de simular la interacción del vehículo con el entorno, para proporcionar una simulación más próxima a la realidad, donde por poner un ejemplo, el dron no cae al comienzo de la simulación por no tenerse en cuenta en su modelo la existencia de una fuerza externa de reacción del suelo. Más detalles sobre estas modificaciones se proporcionan el apartado 9.2. No obstante, todos estos nuevos bloques introducidos están en la actualidad en fase experimental y sería posible en posteriores desarrollos optimizarlos más.

9 VERIFICACIÓN Y VALIDACIÓN DEL MODELO DE SIMULACIÓN

9.1 Planteamiento

Cualquier simulación por ordenador de la dinámica de un proceso físico requiere ser verificada y validada como paso previo a su uso. La verificación permite confirmar que la simulación produce la salida esperada y la validación asegura que dicha salida reproduce fielmente la realidad.

Las tareas de verificación se han realizado de forma iterativa en conjunción con las labores habituales de depuración y corrección de errores que se habían introducido involuntariamente durante la fase de diseño. En este sentido, detrás de los resultados de verificación aquí presentados ha habido una intensa fase de desarrollo y mejora del simulador. El controlador empleado en la fase de verificación ha sido el mismo que se ha descrito en el apartado 7, es decir, una reproducción simulada del desarrollado por el fabricante. Se han realizado una serie de pruebas de verificación a partir de diferentes combinaciones de entradas de referencia y condiciones iniciales. En la evaluación de las pruebas de verificación se ha comprobado que la respuesta del vehículo a las distintas condiciones planteadas es la esperada, que además es estable y que los signos de las variables que forman parte del vector de estado son los que se esperaban en cada momento en base a la señal de referencia y el estado del vehículo.

La validación de la simulación solo se ha podido realizar parcialmente. El motivo ha sido la falta de sensores en el vehículo para medir su posición absoluta en las coordenadas x e y, junto con la falta de exactitud en la estimación de la coordenada z, tal y como se detalló en el apartado 4.2.2. Dado que esta situación se da en la mayoría de las investigaciones de cuadrirotores en interiores, donde no es posible usar la tecnología GPS, lo habitual en estos casos es emplear para esta labor un sistema de captura de movimiento (MOCAP) externo al vehículo que permita estimar la localización del mismo en términos de posición y orientación, desacoplando de esta forma el problema de estimación de estado del problema de control [45] [52] [76]. La principal dificultad para el empleo de dichos sistemas es su elevado coste. Un sistema comercial de captura de movimiento empleado habitualmente para este tipo de evaluaciones, como puede ser el VICON [95], suele estar compuesto de un conjunto de cámaras de infrarrojos con el correspondiente hardware y software asociado que permite establecer con mucha precisión y a una frecuencia alta la localización del vehículo a partir de pequeños marcadores reflectantes instalados en el mismo. El sistema es modular, existiendo distintas configuraciones en función del número y la calidad de las cámaras empleadas y el hardware de soporte, pero incluso para las configuraciones más sencillas tiene un coste de partida por encima de los 50,000 USD (4 cámaras, 5 MP, 420 FPS, 5 x 5 x 5 m de volumen de captura), llegando, en las configuraciones habitualmente empleadas por las universidades, a costes en el entorno de los 200,000-300,000 USD (16 cámaras, 8 MP, 260 FPS, 15 x 15 x 15 m de volumen de captura). Sistemas similares como el OptiTrack [96] tienen un precio inferior, variando también el mismo en función de la configuración elegida (tipo y número de cámaras), entre los 5,000 USD (4 cámaras, 0.3 MP, 100 FPS, 2 x 2 x 2 m de volumen de captura) del más sencillo, a los 150,000 USD del más completo (24 cámaras, 4.1 MP, 180 FPS 15 × 15 × 6 metros de volumen de captura). Asimismo, el fabricante del Crazyflie 2.0 ha estado desarrollando durante el último año una alternativa más económica a los sistemas anteriores basada en radiofrecuencia. La ha denominado sistema de posicionamiento LOCO [97]. El sistema consiste en un pequeño sistema de posicionamiento

local basado en el uso de un cierto número de emisores de radiofrecuencia distribuidos alrededor del área de vuelos (normalmente se precisan más de 6 unidades para obtener buenos resultados), la correspondiente tarjeta hardware y una placa de expansión en el vehículo. El coste del sistema (1,000 EUR) es más asequible que el de los sistemas habitualmente empleados en universidades y centros de investigación. No obstante, el sistema aún está en desarrollo, por lo que no se ha considerado su uso en el presente trabajo. Además, aunque el fabricante ha realizado una primera evaluación de la precisión con la que el sistema es capaz de determinar la localización del Crazyflie 2.0 [98], los resultados son aún preliminares y se necesitaría un estudio más en detalle para confirmar la posibilidad de su uso para tareas de validación de forma alternativa a los sistemas MOCAP, que son suficientemente robustos y conocidos. Por lo tanto, teniendo en cuenta las consideraciones arriba detalladas, en el presente trabajo se ha optado por realizar una validación parcial del simulador empleando los datos proporcionados por los sensores del vehículo y la estimación de estado realizada por su microcontrolador. Esto ha limitado la validación al control de orientación, ya que el vehículo no dispone de sensores que permitan estimar la posición.

9.2 Verificación

El objetivo de la fase de verificación consiste en comprobar que el simulador funciona de acuerdo a su diseño, es decir, que produce como salida una dinámica razonable del sistema objeto de estudio. Dado que la dinámica de la planta está condicionada por el controlador que se le haya conectado, en la fase de verificación se ha empleado el controlador implementado por el fabricante en su firmware, o más concretamente, la simulación del mismo desarrollada en el presente trabajo que está descrita en el apartado 7. Las tareas de verificación se han centrado en el estudio de la evolución del vector de estado ante ciertas entradas escalón de las variables de referencia de localización. En esta fase se han evaluado varios aspectos fundamentales de la simulación, como que el signo de la respuesta sea congruente con el sistema de referencia adoptado (descrito en el apartado 6.3), que los subsistemas de primer orden, como puede ser los motores, muestren, efectivamente una dinámica de primer orden, que los subsistemas de orden superior, como puede ser el vehículo, muestren dinámica de segundo orden, y que la simulación sea estable, así como una serie de aspectos cualitativos de la misma.

9.2.1 Comprobación de ecuaciones, valores de parámetros y unidades

Se ha realizado inicialmente, y de forma previa a la ejecución de las distintas simulaciones de vuelos, una comprobación en el simulador de que los aspectos clave de su funcionamiento se habían establecido de acuerdo a los estudios de planta y controlador realizados previamente y que están descritos en los apartados 6 y 7. Dicha tarea se puede considerar, en su esencia, como un control de conformidad respecto a las especificaciones establecidas.

En primer lugar se ha realizado una comprobación cruzada para asegurar que las ecuaciones de la dinámica de la planta, es decir las ecuaciones (5) a (19), y las ecuaciones de las leyes de control del controlador, es decir las ecuaciones (20) a (28), se habían reproducido de forma correcta en los correspondientes bloques del simulador, bien directamente o a través de funciones de MATLAB® definidas por el usuario. En la comprobación se ha prestado especial atención a los signos en las fórmulas. El objeto de dicha comprobación ha consistido en verificar que no se hubiera introducido ningún error tipográfico o de cualquier otro tipo durante la

construcción del modelo. La comprobación se ha realizado por comparación con los valores presentados en las distintas ecuaciones y en las tablas de parámetros.

A continuación se ha comprobado que los parámetros que emplea el simulador, y que están registrados en los archivos `parametros_planta.m` y `parametros_controlador.m`, se correspondían con los seleccionados en los apartados 6.5.1,6.5.2, 6.5.3 y en el ANEXO 2.

Finalmente se ha realizado una verificación de unidades para asegurar que las unidades empleadas en los distintos submódulos del simulador son congruentes y se corresponden con las usadas en las fórmulas. Especial atención han requerido en este caso los ángulos, ya que debido a que firmware del controlador diseñado por el fabricante trabaja en grados sexagesimales, ha habido que asegurar que las conversiones de radianes a grados y de grados a radianes se realizaban en los lugares oportunos y de forma correcta. En general, el simulador trabaja con radianes, pero los convierte a grados para alimentar los controladores PID y para mostrar los resultados en el postprocesado.

En las comprobaciones anteriormente mencionadas se han descubierto algunos errores que se han corregido en esta fase de depuración.

9.2.2 Casos de prueba

Para verificar el funcionamiento del modelo de simulación se ha diseñado una matriz de verificación que incluye distintas pruebas con su correspondiente referencia (Vxx). Los parámetros que se han empleado en cada una de las pruebas se reproducen en la Tabla 20. Dichos parámetros se han introducido para cada una de ellas de forma manual en la función `parámetros_entorno.m` de forma previa al lanzamiento de cada una de las simulaciones. Se trata de las condiciones iniciales y de las señales de referencia seleccionadas, y que se aplican al vehículo durante la simulación.

Tabla 20 : Pruebas de verificación modelo simulación

	Unidad	V01	V02	V03	V04	V05	V06	V07	V08	V09	V10	V11
x_d	m	0	0	0	1	0	1	-0.1	-0.1	0	0	1
$t(x_d)$	s	0	0	0	20	0	20	20	40	0	0	20
y_d	m	0	0	0	0	1	1	0	-1	0	0	1
$t(y_d)$	s	0	0	0	0	10	20	0	40	0	0	20
z_d	m	1	1	1	1	1	1	1	1	0.2	0.2	1
$t(z_d)$	s	20	10	10	10	60	10	10	20	0	0	20
ψ_d	º	0	90	270	0	0	0	0	0	0	0	315
$t(\psi_d)$	s	0	20	20	0	0	0	0	0	0	0	20
x_0	m	0	0	0	0	0	0	0	0	1	-1	0
y_0	m	0	0	0	0	0	0	0	0	1	-1	0
z_0	m	0	0	0	0	0	0	0	0	1	1	0.2
ϕ_0	º	0	0	0	0	0	0	0	0	0	1	1
θ_0	º	0	0	0	0	0	0	0	0	0	-1	-1
ψ_0	º	0	0	0	0	0	0	0	0	0	45	10
z_1	m	0	0	0	0	0	0	0	n/a	0	0	0

Se ha optado por emplear señales de referencia en escalón en todos los casos, seleccionando el momento de su introducción de forma conveniente para poder verificar el correcto funcionamiento de la simulación en los distintos casos. A pesar de que no es posible verificar todas y cada una de las características dinámicas del modelo se simulación, se ha intentado cubrir un abanico amplio de casos siguiendo la técnica habitual de exploración de la envolvente de vuelo, tratando de plantear casos que mostraran la mayoría de los aspectos dinámicos de interés. Se trata de una primera aproximación a la evaluación de la simulación, pero que ha proporcionado resultados satisfactorios (tras las necesarias correcciones de errores y depuraciones previas) en los aspectos de estabilidad y valores característicos por lo que se puede considerar que el prototipo de simulador elaborado en el presente trabajo ha quedado verificado tras el análisis de los resultados de las simulaciones de verificación planteadas. En el apartado 9.2.3 se reproducen dichos resultados. Se ha decidido realizar simulaciones de 60 segundos en todos los casos, por ser una duración suficiente para comprobar la respuesta transitoria y permanente de la planta, así como la presencia o ausencia de acumulación de errores en la misma.

9.2.3 Resultados obtenidos

Se detallan a continuación los aspectos más relevantes de las distintas pruebas de verificación, mostrando para cada una de ellas la evolución de los distintos componentes del vector de estado a lo largo del tiempo. En el ANEXO 4 se reproducen las gráficas que muestran los parámetros principales relacionados con la planta de potencia (PWM, velocidades de rotación, fuerza y momento resultantes) para cada una de las pruebas.

V01 (despegue y vuelo a punto fijo): iniciamos las pruebas con una simulación del Crazyflie 2.0 en reposo sobre el suelo y le requerimos elevarse a la altura de 1 metro y mantenerse en vuelo a punto fijo en dicha posición. El simulador realiza lo pedido y nos genera la Figura 37, donde podemos observar los valores de los distintos componentes del vector de estado. Todos los componentes son nulos excepto la velocidad vertical que crece desde su valor nulo inicial hasta el sobrepaso, volviéndose negativa por un breve periodo de tiempo en el que el vehículo desciende hasta su posición de velocidad nula en la altura de vuelo a punto fijo seleccionada. El comportamiento tan agresivo se debe a los valores extremadamente altos de las constantes del controlador PID de altura seleccionados por el fabricante. Asimismo, dicho comportamiento se ha presenciado también durante los ensayos en vuelo de validación, donde se ha comprobado que uno de los aspectos más difíciles al comenzar a pilotar el vehículo sin tener experiencia previa en el mismo es el tacto del mando de altura. De hecho es habitual estrellarlo contra el techo en las primeras pruebas.

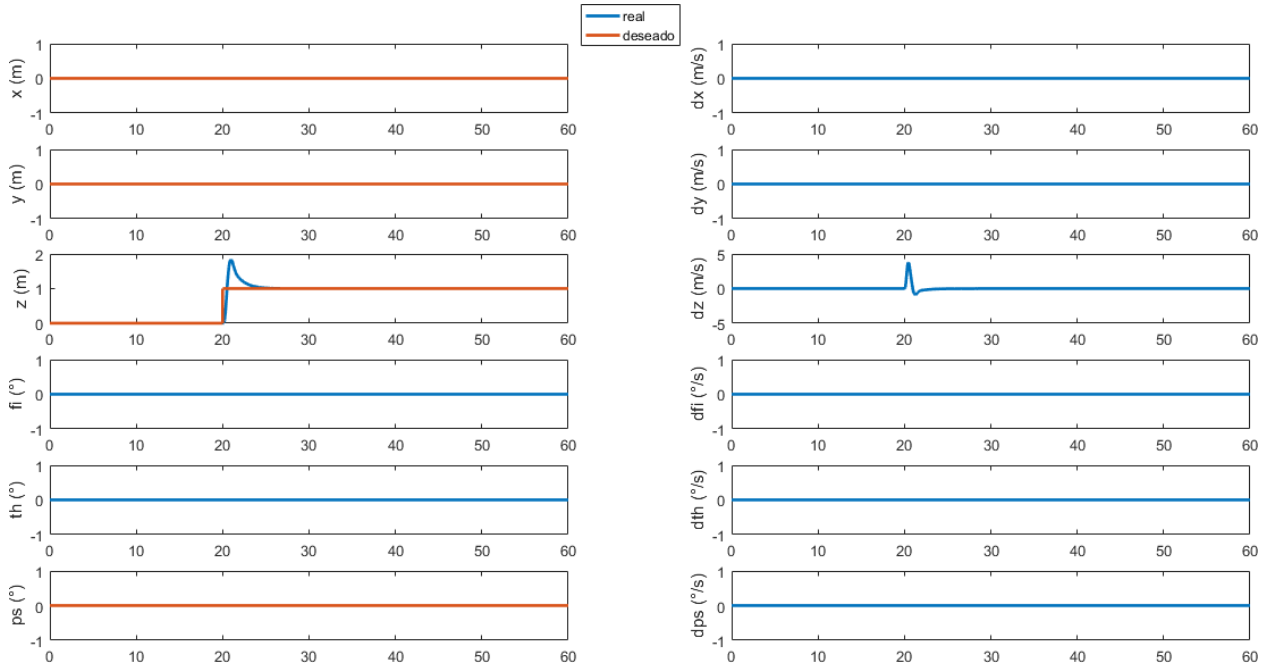


Figura 37 : Despegue y vuelo a punto fijo (V01)

V02 (guiñada): a continuación repetimos la prueba anterior, pero una vez que el vehículo está estabilizado volando a punto fijo le solicitamos girar 90º en el sentido de giro positivo del eje Z. El simulador responde correctamente imprimiendo una velocidad de guiñada anti horaria al vehículo, que tras el sobrepaso se acerca a la posición de 90º de guiñada. Lo podemos ver en la Figura 38.

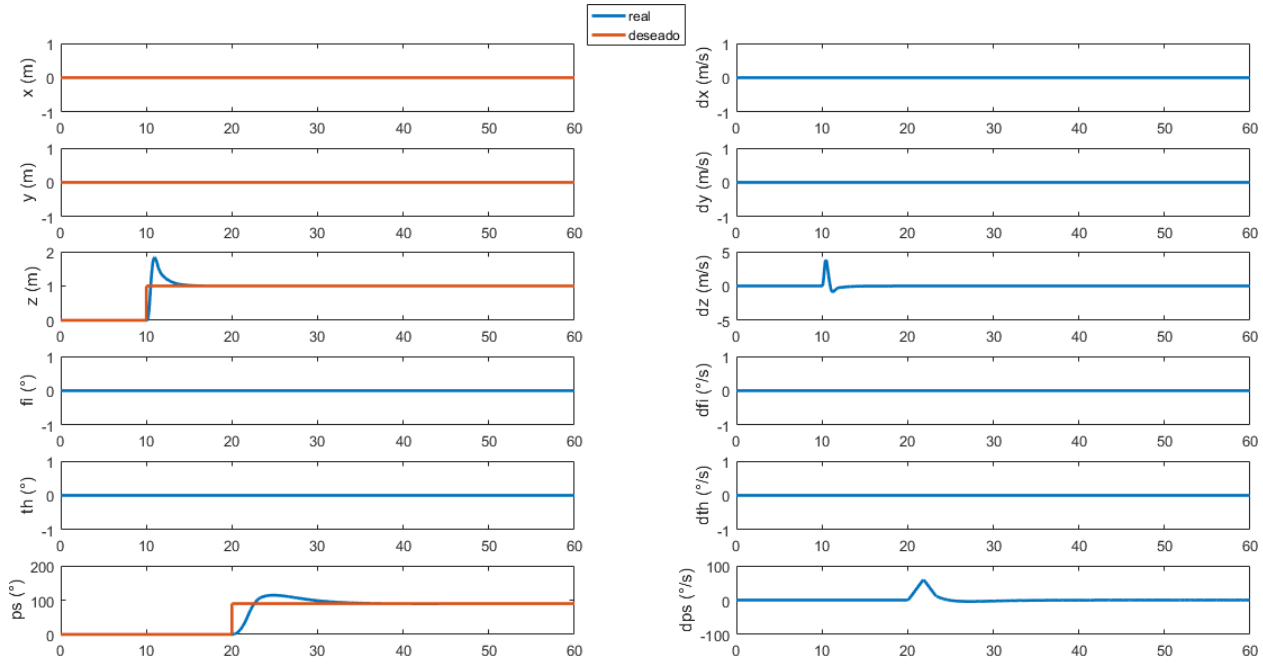


Figura 38 : Guiñada en vuelo a punto fijo (V02)

V03 (limitación ángulo guiñada): una forma de verificar que el simulador realiza correctamente la operación de mantener el ángulo de guiñada en el intervalo ±180º consiste en requerir al vehículo que gire un ángulo de guiñada de 270º a partir de la posición de 0º. Tal y como se ha diseñado el controlador, el vehículo debe realizar una guiñada en el sentido negativo (horaria) para alcanzar el ángulo -90º. En la Figura 39 podemos observar como el vehículo responde correctamente. Nótese que el valor de referencia del ángulo de guiñada introducido es de 270º y que el posprocesador se ha diseñado, asimismo para corregir la referencia, por lo que la figura muestra directamente el valor corregido de -90º.

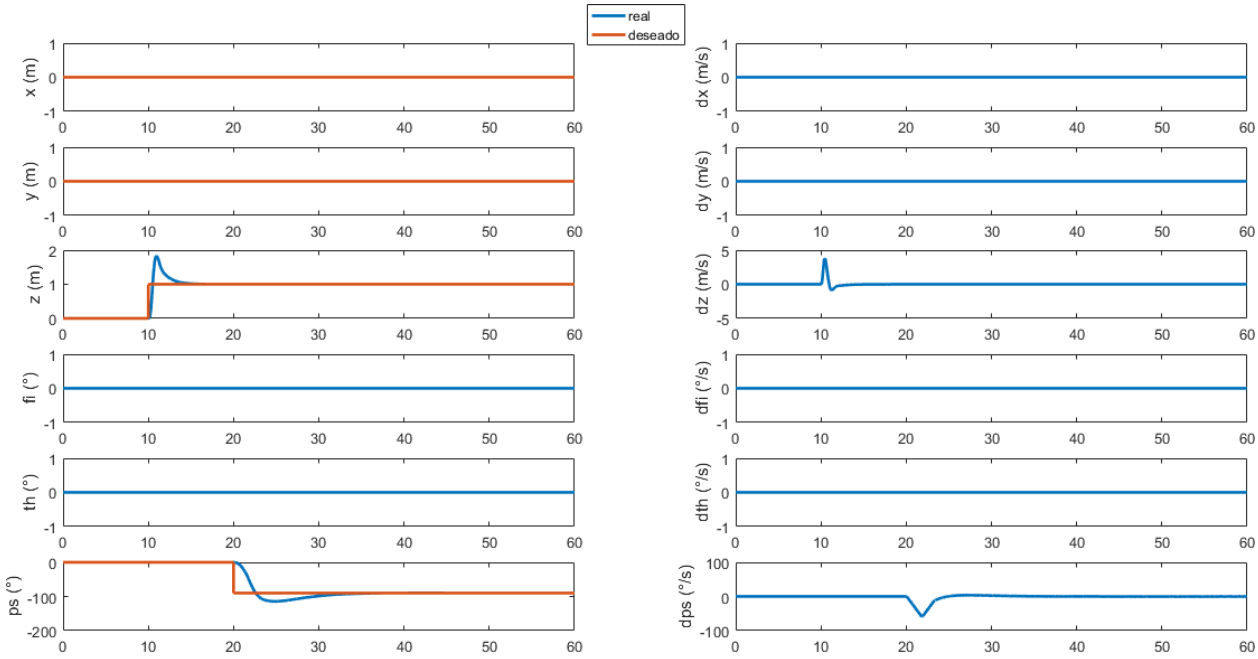


Figura 39 : Guiñada inversa en vuelo a punto fijo (V03)

V04 (desplazamiento horizontal un eje): a continuación fijamos la referencia para vuelo a punto fijo y algunos segundos después requerimos al vehículo moverse 1 m en la dirección positiva del eje X. Podemos observar en la figura Figura 40 que en el simulador se produce un cabeceo positivo de cerca de 1º (alrededor del eje Y) que genera una velocidad horizontal de componente X positiva, tal y como era de esperar.

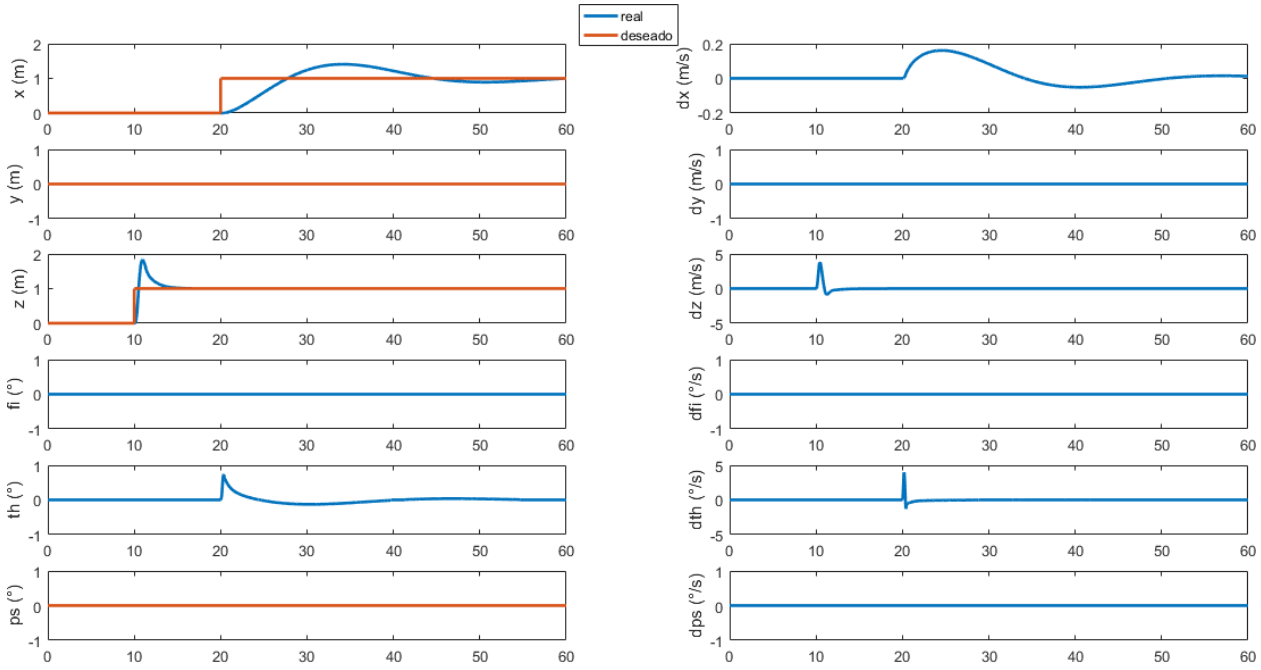


Figura 40 : Vuelo en avance horizontal un eje a partir de punto fijo (V04)

V05 (reseteo del controlador de avance): una de las características introducidas en el simulador es la función de reseteo del control de avance cuando el vehículo está en reposo sobre el suelo y no ha comenzado a volar. Esta función reproduce la que se incluye en el firmware del fabricante. Se ha comprobado su correcto funcionamiento requiriendo al vehículo un desplazamiento horizontal según el eje X cuando está en reposo. Podemos observar en la Figura 41 que la función de reseteo funciona correctamente y no introduce la nueva referencia del eje X hasta el momento en que el vehículo comienza a volar, confirmándose el comportamiento esperado de dicho módulo.

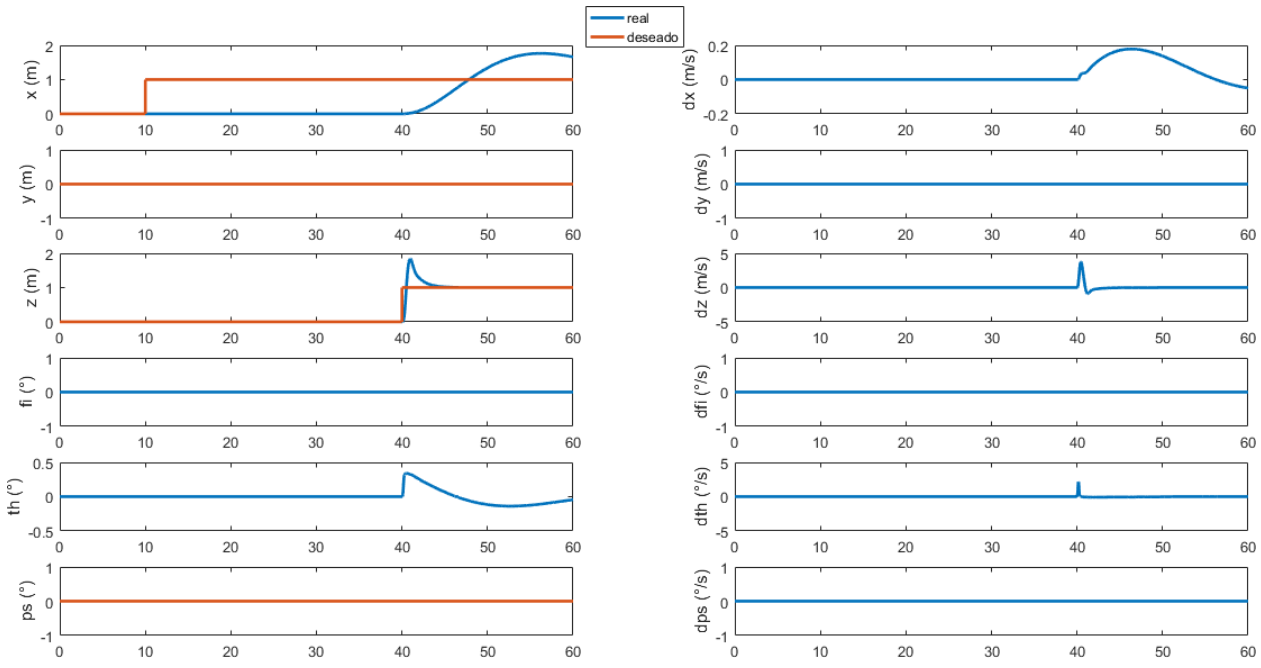


Figura 41 : Vuelo en avance horizontal un eje a partir del despegue (V05)

V06 (desplazamiento horizontal dos ejes): en este caso verificamos que el simulador reproduce correctamente la maniobra de despegue y subida a punto fijo, seguida de un desplazamiento a lo largo de la diagonal del cuadro de 1x1m en el sentido positivo XY. Para ello fijamos en primer lugar la coordenada z deseada en 1m (despegue, subida y vuelo a punto fijo), para tras 10 segundos hacer lo propio con las referencias tanto de X como de Y. En la Figura 42 observamos que los sentidos de los ángulos de alabeo y cabeceo generados por el controlador son los correctos, ya que para realizar esta maniobra el vehículo tiene que simultáneamente alabear en sentido negativo y cabecear en sentido positivo. Estos dos giros simultáneos generan velocidades positivas tanto en el eje X como en el eje Y, con lo que el vehículo realiza el vuelo de avance pretendido.

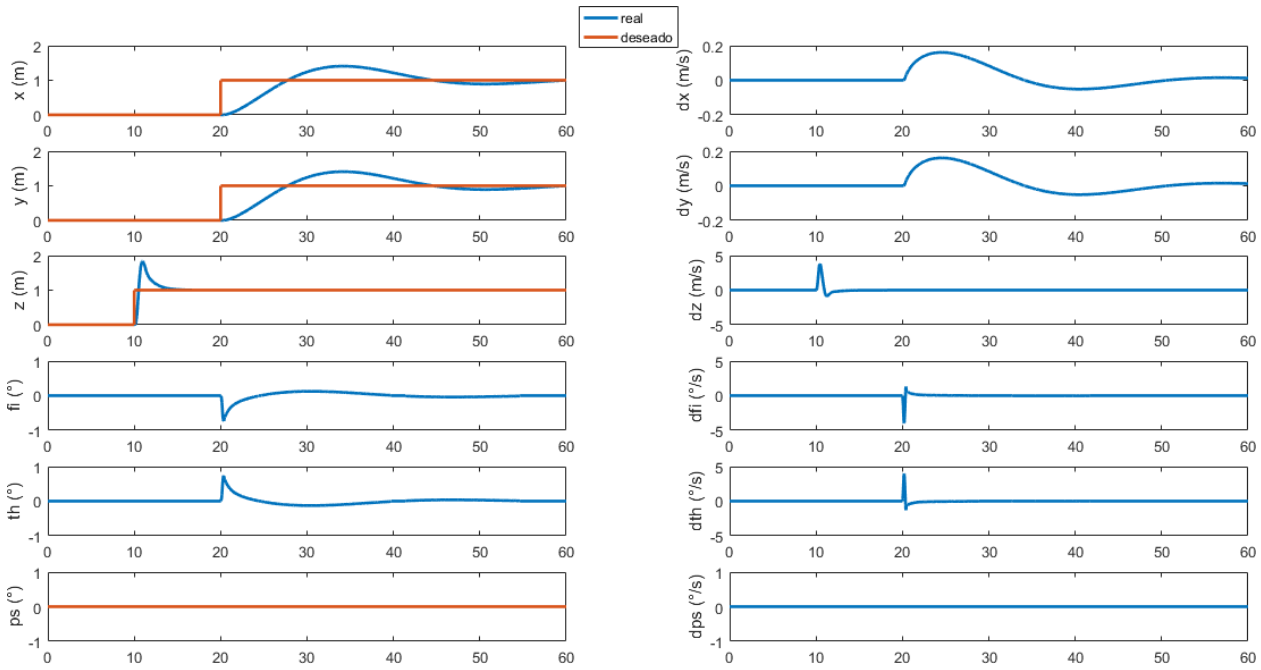


Figura 42 : Vuelo en avance diagonal a partir de punto fijo (V06)

V07 (desplazamiento horizontal corto sentido negativo un eje): a continuación comprobamos simultáneamente la precisión del simulador en recorridos pequeños y en sentido negativo. Para ello requerimos al vehículo que alcance una posición negativa de 10 cm en el eje de las X una vez se ha alcanzado el vuelo a punto fijo. Los resultados se muestran en la Figura 43. Vemos como el simulador realiza un pequeño encabritado del vehículo (que no llega a dos décimas de grado) que le permite imprimir una pequeña velocidad negativa en el eje X con la que puede alcanzar a referencia solicitada, moviéndose 10 cm hacia atrás.

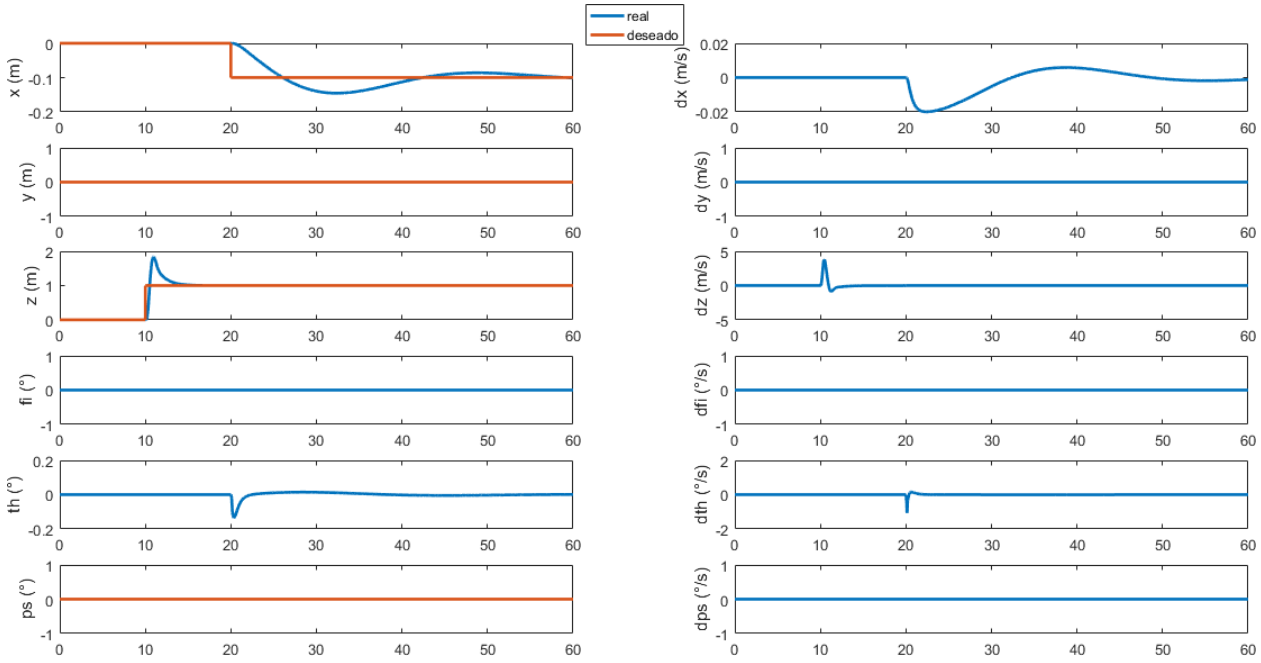


Figura 43 : Vuelo horizontal en retroceso a partir de punto fijo (V07)

V08 (suelo inexistente, desplazamiento horizontal negativo dos ejes no simétrico): en la presente prueba se han realizado dos comprobaciones diferentes. Por un lado se ha deseleccionado la opción de existencia de entorno, por lo que el simulador no tiene en cuenta la presencia del suelo bajo el dron. Asimismo se ha requerido al vehículo alcanzar la altura de 1 m para a continuación desplazarse 10 cm a lo largo del sentido negativo del eje X y 1 m a lo largo del sentido negativo del eje Y de forma simultánea, produciendo, en consecuencia, un desplazamiento diagonal asimétrico. En los resultados de la simulación mostrados en la Figura 44 se puede observar como al comenzar la misma el vehículo empieza descender a valores negativos de Z, ya que no hay suelo que lo sustente, pero el controlador lo sitúa rápidamente en el valor de referencia z= 0 hasta que recibe la señal de mando para situarlo a 1 m, maniobra que ejecuta como en las pruebas anteriores. En cuanto a la segunda comprobación podemos observar que el simulador realiza la maniobra correctamente, comandando el alabeo y cabeceo de forma simultánea y con signos contrarios, siendo el valor del ángulo de cabeceo ejecutado un orden de magnitud inferior al de alabeo. La respuesta del vehículo es la esperada, adquiriendo una velocidad horizontal con componentes negativas en ambos ejes, siendo la del eje X aproximadamente un orden de magnitud inferior a la del eje Y, y alcanzando el punto establecido en la referencia con cierto sobrepaso que el controlador comienza a compensar, pero que no ha terminado completamente al final de la simulación.

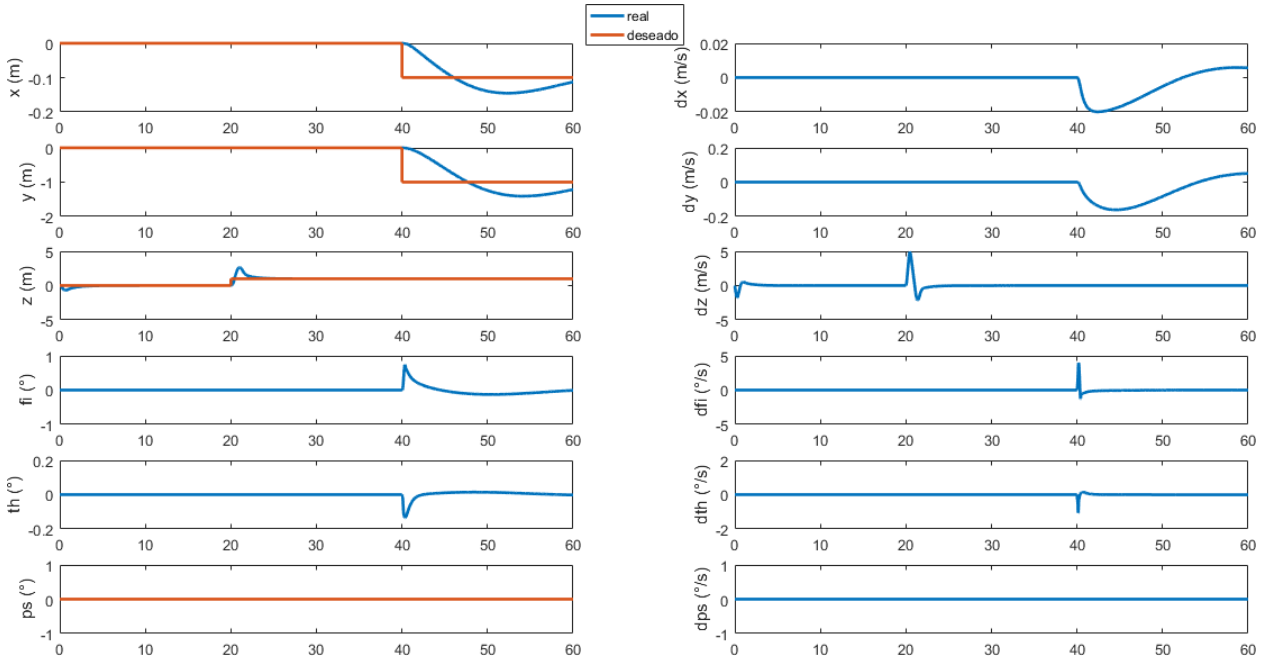


Figura 44 : Despegue sin suelo; vuelo horizontal asimétrico a partir de punto fijo (V08)

V09 (liberación en altura estabilizada con descenso en retroceso y estabilización): en esta prueba se ha liberado el dron en la coordenada (1, 1, 1) en configuración estabilizada, es decir con velocidad nula y ángulos de alabeo, cabeceo y guiñada nulos. Se ha establecido como referencia el punto (0, 0, 0.2), es decir, 20 cm por encima del origen de coordenadas. La respuesta del simulador a la maniobra planteada es la correcta realizando simultáneamente un alabeo positivo y un cabeceo negativo y compensando la velocidad vertical para generar el descenso a la altura de referencia y el retroceso al origen con velocidades negativas en los tres ejes. Se muestra en la Figura 45.

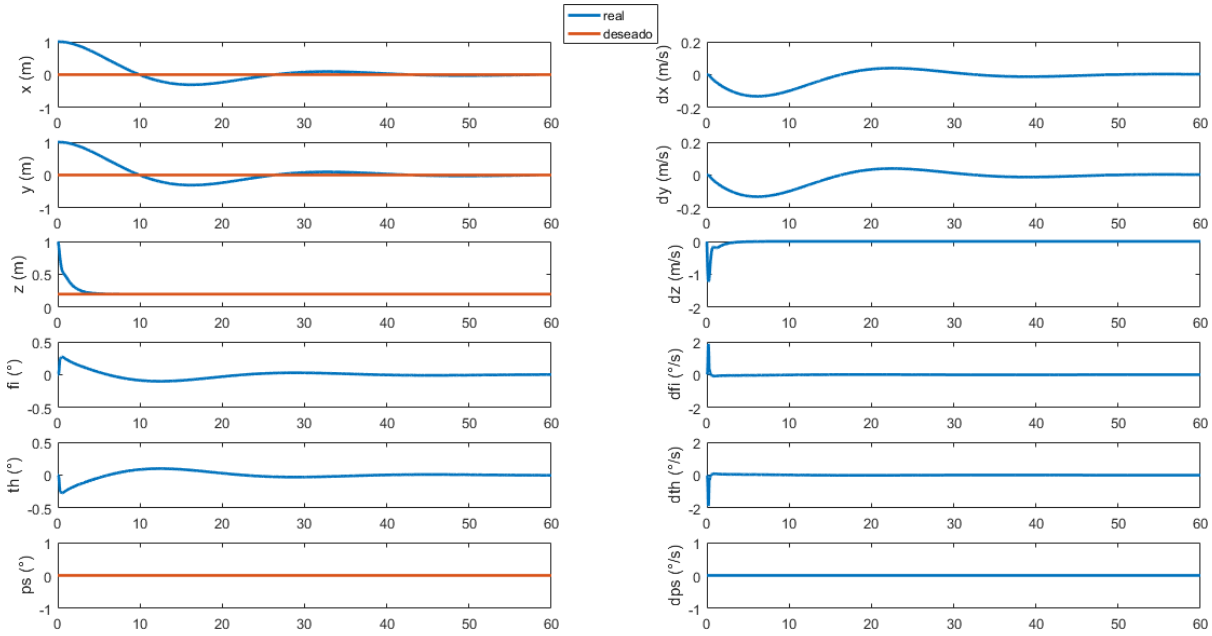


Figura 45 : Liberación en altura estabilizada; descenso con vuelo en retroceso (V09)

V10 (liberación en altura no estabilizada, descenso en avance y rotación): la presente prueba es similar a la anterior, pero en este caso se parte de la coordenada (-1, -1, 1) en configuración no estabilizada. En este caso se parte de un ángulo de alabeo de +1º, de un ángulo de cabeceo de -1º y de un ángulo de guiñada de 45º. Se le requiere al vehículo, al igual que en la prueba anterior descender a la referencia (0, 0, 0.2), para lo cual tiene que comandar la orientación necesaria para el vuelo de avance planteado, que es la contraria a la inicial, al tiempo que controla el descenso y realiza la guiñada de 45º. La respuesta del simulador es la adecuada tanto en lo referente a los ángulos como las velocidades y signos de las mismas, alcanzando el objetivo de forma estabilizada, tal y como se muestra en la Figura 46.

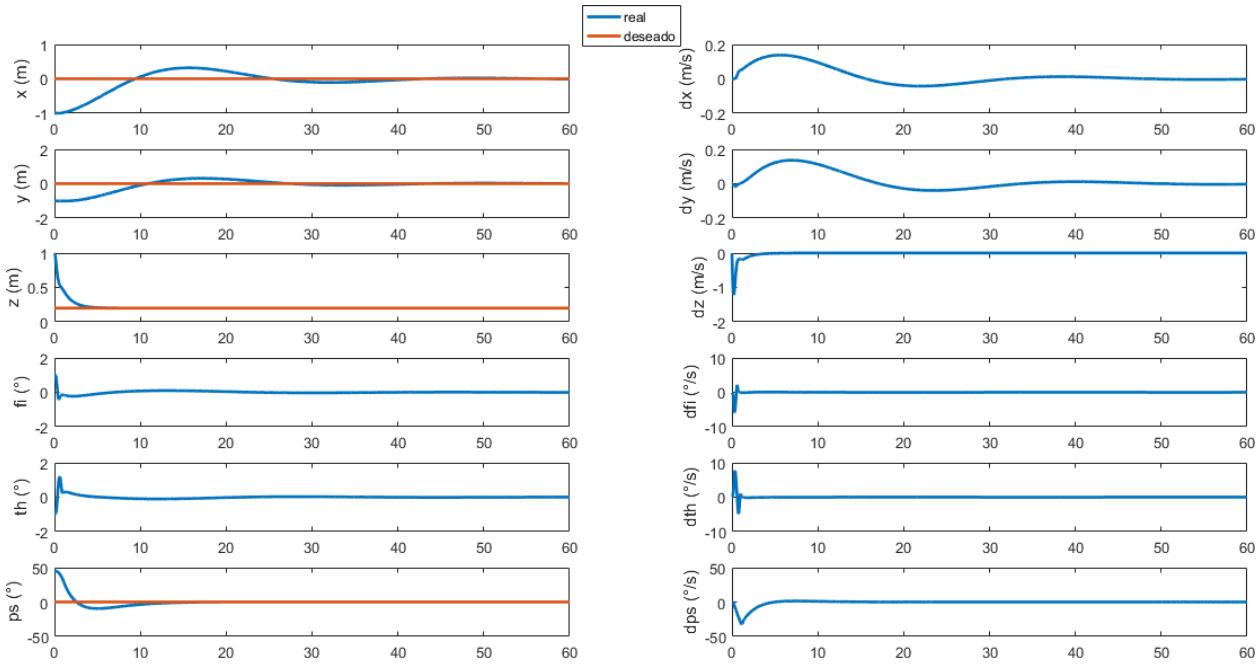


Figura 46 : Liberación en altura no estabilizada; descenso en avance y giro (V10)

V11 (liberación en altura no estabilizada, aterrizaje, despegue en avance y guiñada): en la última prueba del proceso de verificación se han incluido varias maniobras que combinan las anteriormente realizadas. Tenemos al dron no estabilizado a 20 cm de altura sobre el origen de coordenadas y le pedimos que primero aterrice y que después realice simultáneamente un despegue con avance en diagonal de 1 x 1 m y guiñada de 55º. El simulador reproduce la maniobra correctamente, tal y como podemos observar en la Figura 47.

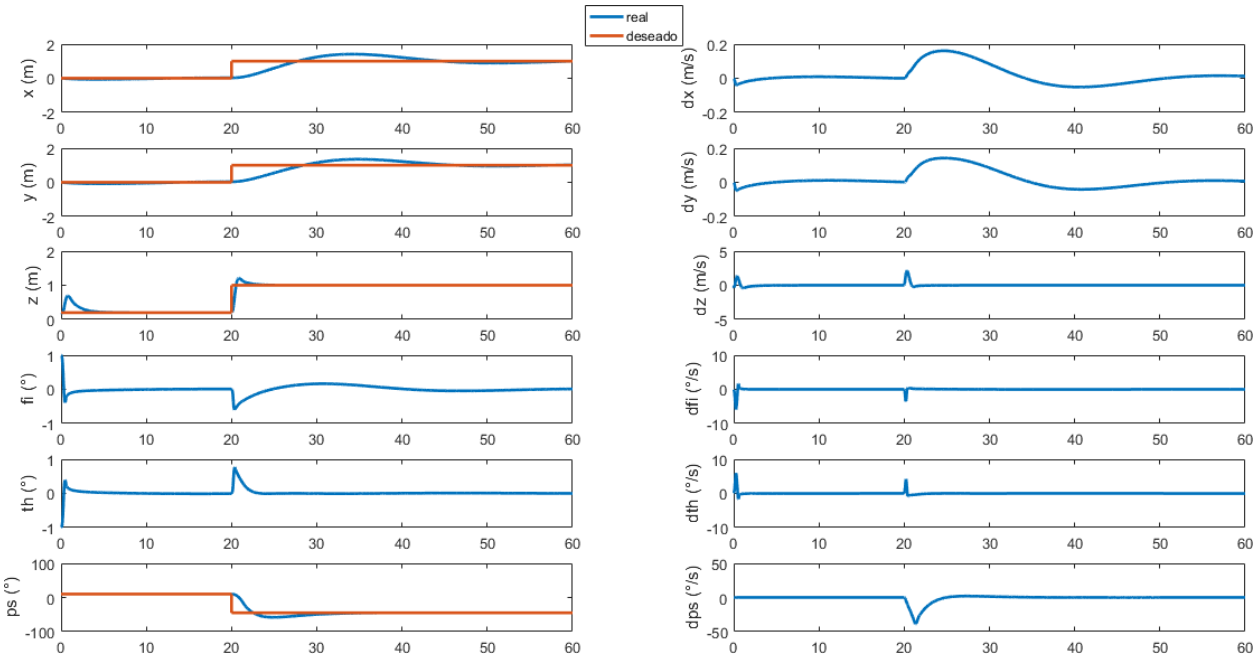


Figura 47 : Liberación en altura; aterrizaje estabilizado; despegue en avance con giro (V11)

9.2.4 Cambios en el modelo de simulación como resultado de la verificación

El proceso de verificación ha sido una de las tareas que más tiempo ha consumido en la presente investigación. Una vez establecido el modelo de simulación se detectaron actuaciones no esperadas en la respuesta dinámica de la planta ante las entradas de referencia que requirieron la identificación de los errores que se habían introducido en la fase de desarrollo y su corrección.

Especial complicación presentaron tanto las unidades de las variables como los signos de las mismas. Estos errores se generaron por desconocer las unidades y criterios de signos que empleaba el controlador implementado por el fabricante. En cuanto a las unidades se siguió un criterio de prueba y error para finalmente identificarlas (emplea una mezcla de unidades del sistema internacional y de otros sistemas), mientras que respecto al criterio de signos se hizo una consulta directamente al fabricante, tal y como se ha descrito en el apartado 4.2.2.

Asimismo, algunos aspectos secundarios de la dinámica del vehículo no se habían tenido en cuenta inicialmente, y fue precisamente la verificación del modelo la que permitió identificarlos y dio pie a su inclusión en el mismo. Como ejemplo se puede citar la ausencia inicial de un subsistema que simulara la presencia del suelo. En las pruebas de verificación se comprobó que al no haberse introducido en el diseño una restricción física respecto a la presencia del suelo, el vehículo podía alcanzar valores de z negativos. Para hacer el simulador más próximo a la realidad, en la que siempre existen restricciones de entorno, se creó un subsistema que simulara el entorno, incluido el suelo, de forma sencilla. En la prueba V08 se muestra el resultado final con el subsistema deshabilitado y en el resto de pruebas con el subsistema habilitado. Otro ejemplo consistió en la ausencia inicial de un subsistema que reseteara el controlador de orientación hasta que se produjera el despegue del vehículo. Esta función la contiene el controlador del fabricante, pero no se había comprendido inicialmente su importancia. Se

introdujo el subsistema correspondiente en el simulador y los resultados se pueden observar en la prueba V05.

Los resultados presentados en el apartado 9.2.3 corresponden a la última versión del simulador en la que los cambios arriba mencionados, junto con muchos otros cuya necesidad se detectó en el proceso de depuración del modelo, están ya implementados, por lo que constituyen el resultado finalmente satisfactorio de las verificación del modelo de simulación. No se han incluido en la presente memoria la gran cantidad de pruebas de desarrollo intermedias con resultado no satisfactorio.

9.3 Validación

La validación del simulador con datos experimentales se ha realizado mediante la comparación entre los datos de vuelo generados durante los vuelos de prueba y los generados por el simulador para las mismas condiciones. Para poder llevar a cabo la campaña de ensayos en vuelo ha habido que habilitar un pequeño entorno diáfano en un sótano. Asimismo, la captura de los datos de vuelo mediante telemetría y su procesamiento posterior ha requerido configurar la estación de tierra para esta tarea y elaborar un código que automatice el procesado de los datos. Por otro lado, se ha tenido que modificar el simulador para poder inyectarle la misma señal de referencia empleada por el microcontrolador del vehículo en cada momento. Todas estas tareas, así como los resultados de la validación se presentan en los apartados siguientes.

9.3.1 Entorno para uso en la campaña de ensayos en vuelo

La realización de los vuelos de prueba requiere de un entorno seguro en el que no existan obstáculos y en el que tanto los impactos de la aeronave con el entorno como sus consecuencias sean minimizados. La grupos de investigación que actualmente cuentan con líneas de investigación en control de UAVs disponen de entornos de vuelos de prueba, también denominados arenas, o test beds [52] [45], de dimensiones adecuadas y en la mayoría de los casos dotados de sistemas de captura del movimiento, como puede ser el sistema comercial Vicon [95], que les permiten realizar convenientemente dichos vuelos de prueba, estimar la posición del vehículo con una alta precisión y desconectar la parte de estimación de estado de la planta de la parte de investigación en nuevos sistemas de control del vehículo. En el caso del presente trabajo no se ha tenido acceso a una arena de esas características, y se ha optado por usar un entorno construido en un sótano, que proporciona las características de vuelo seguro, pero que no cuenta con ningún sistema de captura de movimiento. Esta elección conlleva ciertas limitaciones en la calidad de los datos obtenidos y en las características de la validación que se puede realizar.

Para el desarrollo de los vuelos de prueba se ha liberado un área de 2 x 2 x 2 m en un sótano de una vivienda. Se han protegido los límites del entorno con cortinas para evitar los impactos del vehículo con paredes o elementos rígidos y disminuir así el riesgo de destrucción de alguna de sus piezas del vehículo, como hélices, brazos soporte, etc. En la Figura 48 se muestra el entorno empleado para los vuelos de prueba. Se puede observar el área cúbica empleada para los vuelos, que está protegida por cortinas, así como la estación de tierra para el operador. La generación del modelo 3D del área de vuelos se realizó mediante la herramienta SketchUp Make [99] y fue uno de los trabajos realizados por una estudiante de secundaria de la asignatura de informática encaminados a la elaboración de modelos 3D replicando geometrías existentes.

Figura 48 : Entorno usado en campaña de ensayos en vuelo

9.3.2 Telemetría

La obtención de los datos de vuelo correspondientes a los ensayos de validación se ha realizado mediante telemetría. Para ello se ha necesitado realizar una configuración previa del software de la estación de tierra para, a continuación, seguir con una fase de obtención y tratamiento de datos. El firmware desarrollado por el fabricante permite configurar el microcontrolador para que realice la tarea de captura y registro de datos de vuelo de forma adicional a su labor principal de lectura de sensores y estabilización del vehículo. Para ello es necesario configurar los parámetros que se quieren registrar y su frecuencia de muestreo a través del cliente de la estación de tierra. Una vez realizada la configuración, los datos se transfieren a la estación de tierra a través de la antena del vehículo y el cliente de la estación gestiona su grabación en archivos de datos separados por comas (csv) en tiempo real.

En el presente trabajo se ha configurado el cliente de la estación de tierra para grabar ocho grupos de parámetros de vuelo para su uso posterior en la validación del modelo de simulación. En el ANEXO 5 se listan los parámetros incluidos en cada uno de los grupos y se reproduce, como ejemplo, uno de los archivos de configuración del cliente.

Se han seleccionado los grupos de parámetros que eran necesarios para la validación. Alguno de ellos, como puede ser el grupo LOC2 en el que se registra la posición deseada, no se ha usado, por las razones explicadas en el apartado 9.1, pero se han construido las necesarias provisiones en el simulador y en la configuración del cliente para poder usarlo si se dispone de un sistema MOCAP que se pueda emplear para la validación.

9.3.3 Procesamiento de datos

Para el procesamiento de los datos de vuelo obtenidos por telemetría se ha creado una herramienta de tratamiento de datos (`Crazytelemetria.m`) escrita para MATLAB®. La herramienta se ha desarrollado para automatizar el monótono procesado manual de los datos de vuelo grabados por el cliente durante los vuelos de prueba que es necesario realizar antes de su visualización y estudio. De esta forma se garantiza la calidad de los datos evitando la posibilidad de introducir errores humanos debidos al procesamiento manual de los mismos. En el ANEXO 6 se reproduce el código de la herramienta de tratamiento de datos que se ha escrito para realizar dicha tarea, así como el código de los archivos auxiliares necesarios para la conversión de formato `.csv` a estructuras de datos.

La herramienta incluye en la cabecera las instrucciones necesarias para su uso. Una vez obtenidos los datos de vuelo, tras la correspondiente prueba en vuelo, se copia la carpeta generada por el cliente, que contiene los 8 archivos de datos en formato `.csv`, en una ubicación elegida por el usuario. Tras introducir la ruta de dicha carpeta en la herramienta y ejecutar la misma, se genera de forma automatizada una estructura que contiene todos los datos de vuelo obtenidos y que se guarda para su uso posterior en un único archivo, al que la herramienta da por nombre la fecha y hora del ensayo. La herramienta permite realizar el procesado de varios vuelos a la vez de forma que se puede realizar una campaña de vuelos, en la que el cliente de la estación de tierra generará una carpeta por vuelo, y más tarde procesar todas las carpetas al mismo tiempo, copiándolas en una carpeta temporal y ejecutando la herramienta `Crazytelemetria.m`. En este caso la herramienta generará un archivo `.mat` por cada uno de los vuelos de la campaña.

La herramienta no solo procesa los datos, sino que realiza su tratamiento, ajustando las unidades, si es necesario, para su uso por el modelo de simulación o para su representación gráfica, ya que los datos de vuelo obtenidos se emplean tanto en el preprocesado, para proporcionarlos como entrada al modelo de simulación, como en el posprocesado, para representarlos junto a los procedentes de la simulación.

El acceso a los distintos parámetros guardados en los archivos de datos de vuelo se realiza a través de la correspondiente ruta, ya que el archivo `.mat` de un determinado vuelo de prueba contiene una estructura de datos en la que se han registrado todos los parámetros del vuelo. La ruta consta de cuatro elementos. En la Tabla 21 se muestra un ejemplo de ruta y de los distintos componentes de la misma. En este caso se trata del ángulo de alabeo deseado a lo largo del tiempo que queda registrado en dicho elemento de la estructura de datos como serie temporal.

Tabla 21 : Ejemplo ruta acceso datos de vuelo

Ruta de acceso parámetro N. 01	`telemetria.REF.ctrltarget.roll`	
Elementos ruta	Palabra fija	`telemetria`
	Grupo captura (definido en cliente estación)	`REF`
	Grupo registro (definido en firmware)	`ctrltarget`
	Nombre variable (definido en firmware)	`roll`

En la Tabla 22 se muestra un listado de los cuarenta y un (41) parámetros que se registran en cada vuelo con su correspondiente ruta de acceso y las unidades en las que se registran.

Tabla 22 : Listado de parámetros de vuelo registrados

N.	Ruta	Parámetro (origen)	Unidades
01	telemetria.REF.ctrltarget.roll	Ángulo alabeo deseado (mando)	º
02	telemetria.REF.ctrltarget.pitch	Ángulo cabeceo deseado (mando)	º
03	telemetria.REF.ctrltarget.yaw	Ángulo guiñada deseado (mando)	º
04	telemetria.REF.sys.canfly	Cargador desconectado y batería ok	0,1
05	telemetria.SEN1.acc.x	Aceleración eje X (sensor)	g
06	telemetria.SEN1.acc.y	Aceleración eje Y (sensor)	g
07	telemetria.SEN1.acc.z	Aceleración eje Z (sensor)	g
08	telemetria.SEN1.baro.asl	Altitud (sensor)	m
09	telemetria.SEN1.baro.temp	Temperatura (sensor)	ºC
10	telemetria.SEN1.baro.pressure	Presión (sensor)	hPa
11	telemetria.SEN2.gyro.x	Velocidad giro eje X (sensor)	º/s
12	telemetria.SEN2.gyro.y	Velocidad giro eje Y (sensor)	º/s
13	telemetria.SEN2.gyro.z	Velocidad giro eje Z (sensor)	º/s
14	telemetria.SEN2.mag.x	Orientación magnética eje X (sensor)	º
15	telemetria.SEN2.mag.y	Orientación magnética eje Y (sensor)	º
16	telemetria.SEN2.mag.z	Orientación magnética eje Z (sensor)	º
17	telemetria.STA.stabilizer.thrust	Empuje estimado (control)	uint16
18	telemetria.STA.stabilizer.roll	Ángulo alabeo estimado (control)	º
19	telemetria.STA.stabilizer.pitch	Ángulo cabeceo estimado (control)	º
20	telemetria.STA.stabilizer.yaw	Ángulo guiñada estimado (control)	º
21	telemetria.LOC1.contoller.actuatorThrust	U1 deseado (control)	uint16
22	telemetria.LOC1.contoller.roll	U2 deseado (control)	º
23	telemetria.LOC1.contoller.pitch	U3 deseado (control)	º
24	telemetria.LOC1.contoller.ctr_yaw	U4 deseado (control)	uint16
25	telemetria.LOC1.contoller.yaw	U4 deseado advanced mode (control)	º
26	telemetria.LOC1.posEstimatorAlt.estimatedZ	Altitud estimada (control)	m
27	telemetria.LOC1.posEstimatorAlt.velocityZ	Velocidad vertical estimada (control)	m/s
28	telemetria.LOC2.posCtlAlt.targetX	No usado	-
29	telemetria.LOC2.posCtlAlt.targetY	No usado	-
30	telemetria.LOC2.posCtlAlt.targetZ	No usado	-
31	telemetria.LOC2.posCtlAlt.p	No usado	-
32	telemetria.LOC2.posCtlAlt.i	No usado	-
33	telemetria.LOC2.posCtlAlt.d	No usado	-
34	telemetria.MOT1.motor.m1	Señal control motor 1 (control)	uint16
35	telemetria.MOT1.motor.m2	Señal control motor 2 (control)	uint16
36	telemetria.MOT1.motor.m3	Señal control motor 3 (control)	uint16
37	telemetria.MOT1.motor.m4	Señal control motor 4 (control)	uint16
38	telemetria.MOT2.pwm.m1_pwm	PWM motor 1 (control)	0-1
39	telemetria.MOT2.pwm.m2_pwm	PWM motor 2 (control)	0-1
40	telemetria.MOT2.pwm.m3_pwm	PWM motor 3 (control)	0-1
41	telemetria.MOT2.pwm.m4_pwm	PWM motor 4 (control)	0-1

9.3.4 Modificaciones del simulador para inyección de señales

El simulador inicialmente desarrollado solo permitía fijar la señal de referencia de posición y ángulo de guiñada del vehículo a lo largo del tiempo. Dicha señal se configuraba en el archivo `configuracion__simulador.m`, dado que para la fase de verificación únicamente se precisaban señales de referencia en escalón que se pueden generar fácilmente a partir de dos parámetros. El simulador era capaz de funcionar correctamente y simular la dinámica del vehículo tal y como se vio en el apartado 9.2. Tanto para completar la fase de validación, como para usar posteriormente el simulador en otras aplicaciones, es necesario contar con la posibilidad de inyectarle señales, bien obtenidas directamente de la realidad, a través de sensores, o bien generadas sintéticamente, con objeto de poder comparar datos de vuelo reales o sintéticos por los generados por el simulador. En las actividades correspondientes a la validación del simulador se han introducido algunos puntos de entrada de señal controlados mediante interruptores (switches) para futuros desarrollos y usos. Los puntos de inyección se controlan a través del archivo de configuración del simulador.

En el archivo de configuración del simulador se ha introducido el concepto de vector de configuración que permite configurar de una forma cómoda el punto de inyección de las señales, o bien seleccionar no inyectar ninguna señal y trabajar en modo de simulación. Las instrucciones acerca de la forma de establecer la configuración del simulador y de fijar la ruta del archivo de datos de vuelo a emplear se incluyen en el propio archivo, que se ha reproducido en el ANEXO 3.

En la Figura 49 se puede observar uno de los puntos de inyección introducidos en el modelo de simulación. A través de él, y una vez seleccionada la opción en el archivo de configuración del simulador, se pueden introducir los parámetros de vuelo 22, 23 y 24, en vez de los correspondientes al controlador.

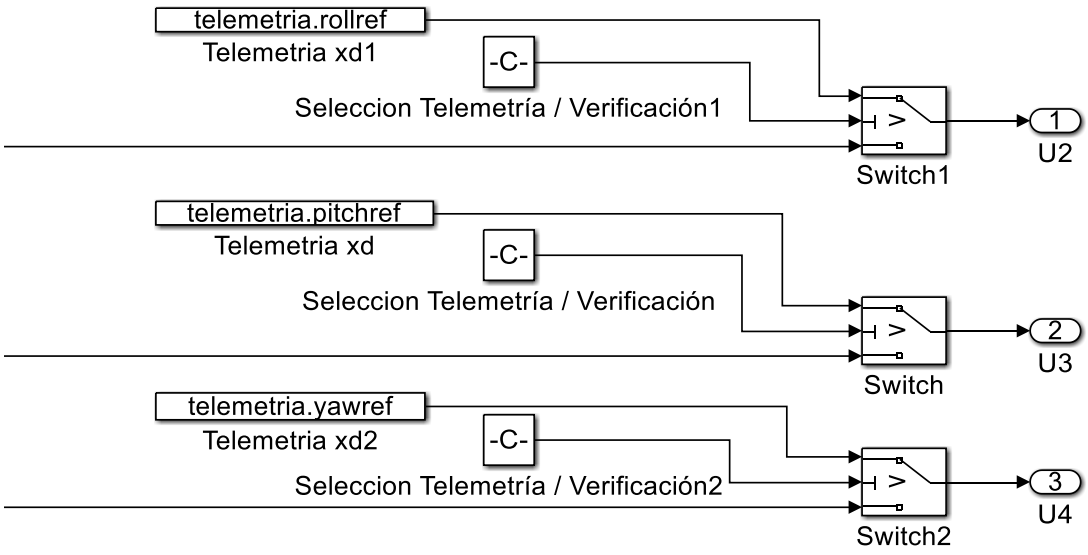


Figura 49 : Punto de inyección de señal externa

Los puntos de inyección de señal se han introducido en distintos bloques del simulador, para uso presente y futuro. Existe un punto en la estación de tierra que permite inyectar las señales de un sistema MOCAP a través de los parámetros de vuelo 28-33. Este punto no se ha usado en el presente trabajo por las razones detalladas en los apartados 9.1 y 9.3.6. Asimismo, existe otro punto de inyección en el bloque del controlador de localización que permite evaluar la interface con el controlador de los motores. Finalmente, se ha introducido otro punto de inyección en la interface entre el controlador y la planta que permite inyectar los valores de los voltajes de mando de los actuadores obtenidos a través de datos de vuelo.

9.3.5 Campaña de ensayos en vuelo

Se ha realizado un cierto número de vuelos de prueba con el Crazyflie 2.0 en el entorno descrito en el apartado 9.3.1. La campaña de ensayos en vuelo no ha presentado ningún problema, aparte de los habituales de impactos con el entorno y rotura de elementos del vehículo. En nuestro caso se fracturó uno de los soportes de los motores al impactar el Crazyflie 2.0 a gran velocidad contra el techo, tal y como se muestra en la Figura 50.

Figura 50 : Aspecto del Crazyflie 2.0 tras un impacto contra el entorno

Durante la campaña de ensayos, se han obtenido los parámetros de vuelo mediante telemetría, tal y como se ha detallado en el apartado 9.3.2, y se han procesado según se explicó en el apartado 9.3.3.

En la Tabla 23 se recogen los principales datos de los diferentes vuelos realizados en la campaña. Para cada vuelo se lista el nombre del archivo de datos de vuelo una vez procesado, la duración aproximada del vuelo y una descripción muy somera del mismo. La descripción es importante de cara a la evaluación de los datos de vuelo. No obstante, los vuelos también fueron registrados y se cuenta con un archivo de video del mismo nombre del archivo de datos de vuelo para cada uno de ellos. Los archivos de video se pueden consultar a la hora de interpretar los resultados de los datos de vuelo, junto con los comentarios, que muestran las impresiones del operador de cada vuelo sobre el desarrollo del mismo.

Es importante destacar que el Crazyflie 2.0 ha sido operado durante los ensayos en vuelo por el autor del presente trabajo, que no poseía ninguna experiencia previa sobre el pilotaje de drones excepto el conocimiento necesario de las fuerzas y momentos involucrados en el proceso. Como consecuencia de dicha falta de experiencia, ha sido difícil realizar vuelos estables con el vehículo en la campaña de ensayos. Además, el vehículo no parece ser fácil de pilotar si no se posee experiencia previa en dicha actividad, ya que al igual que el resto de los drones de la categoría nano se trata de un vehículo muy ágil en el que se alcanzan velocidades elevadas, tanto lineales como angulares, de forma habitual. Por otro lado, el fabricante del vehículo está continuamente tratando de mejorar la controlabilidad del vehículo, a través de mejoras en el firmware, ya que es consciente del problema mencionado. De hecho, parece que los últimos cambios del mismo, introducidos muy recientemente, han mejorado sustancialmente este aspecto.

Tabla 23 : Ensayos en vuelo realizados

Ref	Fichero (.mat)	t (s)	Comentarios
P01	20170531T14-55-22	19	Subida estable seguida de pequeña caída y desestabilización.
P02	20170531T15-08-36	10	Subida inestable, vuelo no controlado.
P03	20170531T15-26-04	10	Subida estable seguida de separación de una hélice en vuelo e impacto contra suelo
P04	20170531T15-36-04	15	Subida inestable con movimiento horizontal. Impacto final contra el suelo con rebote
P05	20170531T15-40-36	26	Vuelo estable con impacto final contra el suelo
P06	20170531T15-46-49	17	Vuelo inestable en altura con varios impactos contra el suelo
P07	20170531T15-51-02	18	Vuelo inestable en altura con un pequeño impacto en suelo antes del impacto final
P08	20170531T15-56-08	23	Vuelo algo inestable en altura con dos impactos en suelo antes de impacto final.
P09	20170531T16-00-50	13	Vuelo muy inestable, con cuatro impactos contra el suelo.
P10	20170603T12-31-45	10	Subida, bajada hasta tocar el suelo. Dos rebotes, subida y bajada final
P11	20170603T13-05-02	16	Subida y estabilización. Vuelos circulares y bajada
P12	20170603T13-09-59	10	Despegue inestable y movimiento horizontal rápido hasta el impacto contra elsuelo
P13	20170603T13-16-26	18	Inestabilidad en altura
P14	20170603T13-20-54	19	Estable en altura, inestable horizontalmente, giro
P15	20170603T13-25-15	23	Subida estable y vuelo estable. Prueba de movimiento horizontal

9.3.6 Resultados obtenidos

Para el uso de los datos de vuelo obtenidos en la campaña se ha introducido en el preprocesador una herramienta muy básica de selección de la ventana temporal a tratar. Esto permite elegir el intervalo de tiempo del que queremos mostrar los datos de vuelo. Asimismo, se permite simular únicamente una parte de un vuelo de prueba, lo que ayuda en la evaluación de los resultados, dado que se puede eliminar la parte de vuelo inestable, o el choque del vehículo con el entorno, o incluso un largo periodo en el que el vehículo se encuentra en el suelo antes de despegar. Así, cuando hemos configurado el simulador en modo validación con un vuelo de prueba concreto, y contamos con el archivo de datos de vuelo correspondiente, podemos ejecutar el mencionado simulador, y se nos presentarán los parámetros principales del vuelo completo en una ventana similar a la mostrada en la Figura 51.

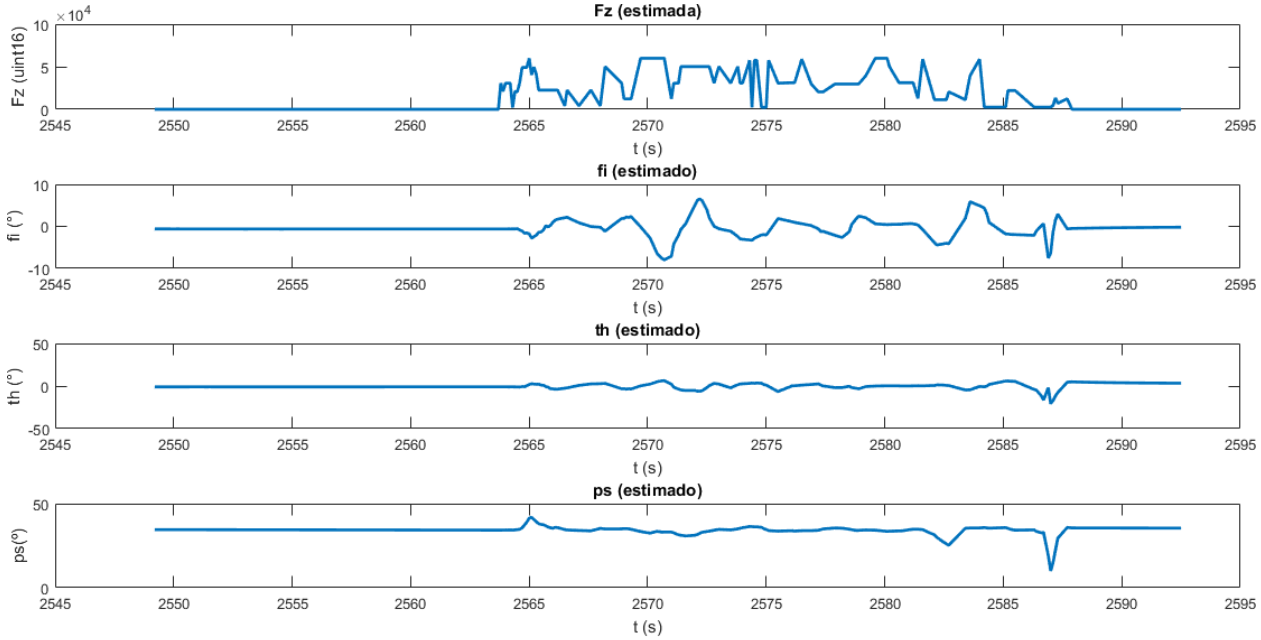


Figura 51 : Ventana de selección, vuelo P05 (20170531T15-40-36)

A continuación se nos pide que seleccionemos el tiempo inicial y el final que vamos a usar en la validación. Una vez hecho esto, el simulador elimina el resto de datos de vuelo y lanza la simulación de la ventana de tiempo seleccionada. Tras ello muestra los datos de vuelo pertinentes y los compara con los datos generados por la simulación para el periodo de tiempo seleccionado, produciendo una serie de figuras, que se analizan a continuación.

En el caso del vuelo P05, mostrado en la Figura 51, se selecciona el periodo de tiempo (2560, 2590) con objeto de analizar únicamente la fase de vuelo, ya que la parte en la que el vehículo está parado en el suelo no nos interesa. Una vez hecho esto, el simulador lanza la simulación y al finalizar nos muestra, en primer lugar los datos de vuelo más relevantes para la ventana de tiempo seleccionada, seguidos por los datos de la simulación. A continuación se van a proporcionar detalles sobre la información que presenta el simulador para el sector de vuelo antes mencionado.

Así, en la Figura 51 se muestran los valores de PWM suministrados a cada uno de los actuadores del vehículo tal y como han sido capturados por los sensores del mismo. Para simplificar se han representado los valores de PWM en tanto por uno, en vez del valor real, que es en porcentaje. Podemos observar que los voltajes tienen una alta variabilidad a lo largo del tiempo, presentando un perfil en 'sierra' de forma muy consistente. Este hecho, que se corresponde con un control en tiempo discreto, como es el que implementa el controlador del fabricante del vehículo, crea problemas a la hora de inyectar dicha señal en el simulador, que está basado en un modelo en tiempo continuo. Como se mostrará más adelante, la inyección de los valores de los voltajes aplicados a los actuadores directamente en la simulación genera inestabilidad, no pudiéndose realizar en dicho caso una comparativa adecuada.

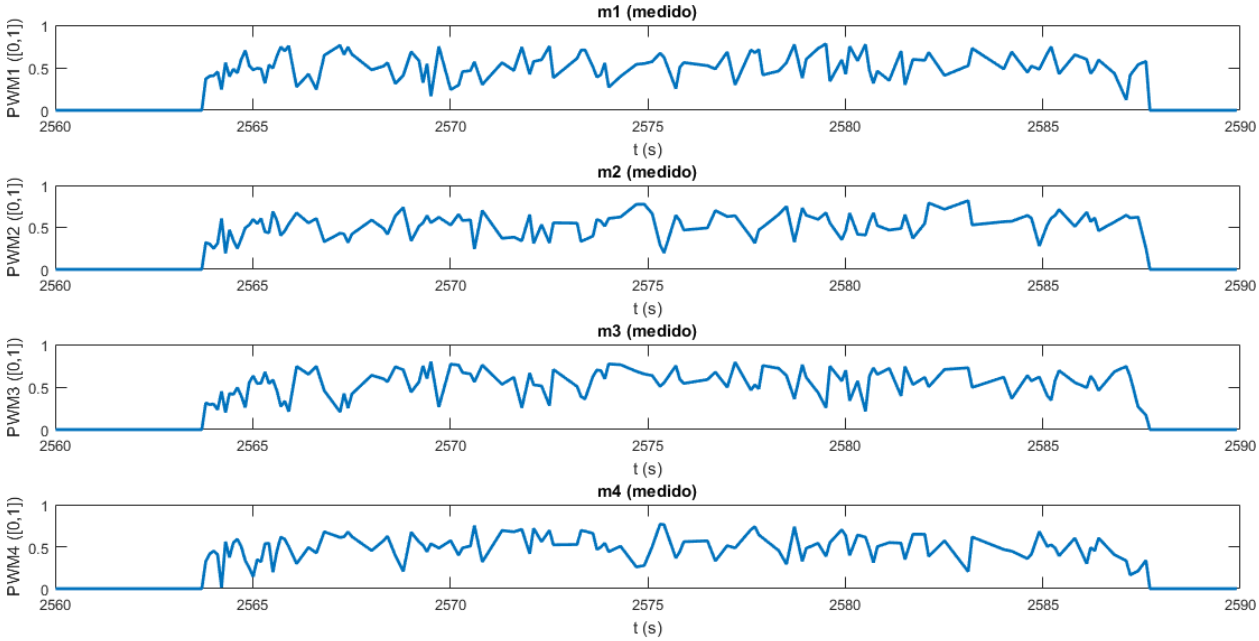


Figura 52 : PWM, vuelo P05 (20170531T15-40-36)

También es interesante observar una comparativa entre los valores de referencia introducidos en el mando por el operador de vuelo y la señal de control generada por el controlador para reproducir la trayectoria comandada. Dicha comparativa se muestra en el caso de la orientación del vehículo en la Figura 53. En azul se indican los ángulos deseados, mientras que en rojo se reproduce la señal del control que el controlador de localización genera y envía al controlador de la planta de potencia. En el caso presente se observa que el controlador reproduce las señales de referencia correctamente.

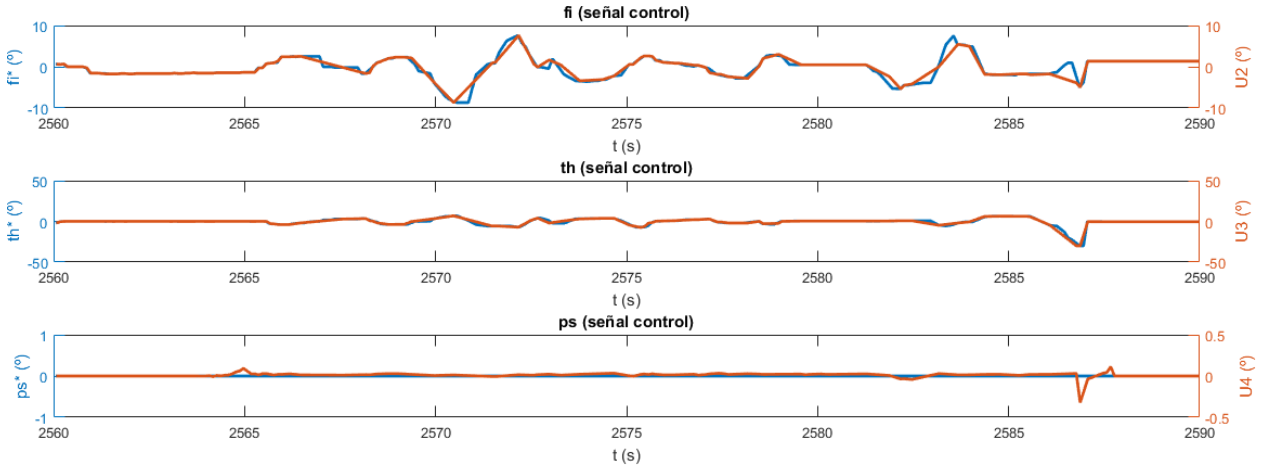


Figura 53 : Control orientación, vuelo P05 (20170531T15-40-36)

En la siguiente figura se puede observar el efecto que la señal de control tiene en la orientación de la aeronave, presentándose la evolución temporal de los ángulos de Euler, tal y como es calculada por fusión sensorial por el firmware desarrollado por el fabricante. Así, en la Figura 54 podemos ver dicha evolución y cómo se corresponde correctamente con la imprimida por la señal del controlador.

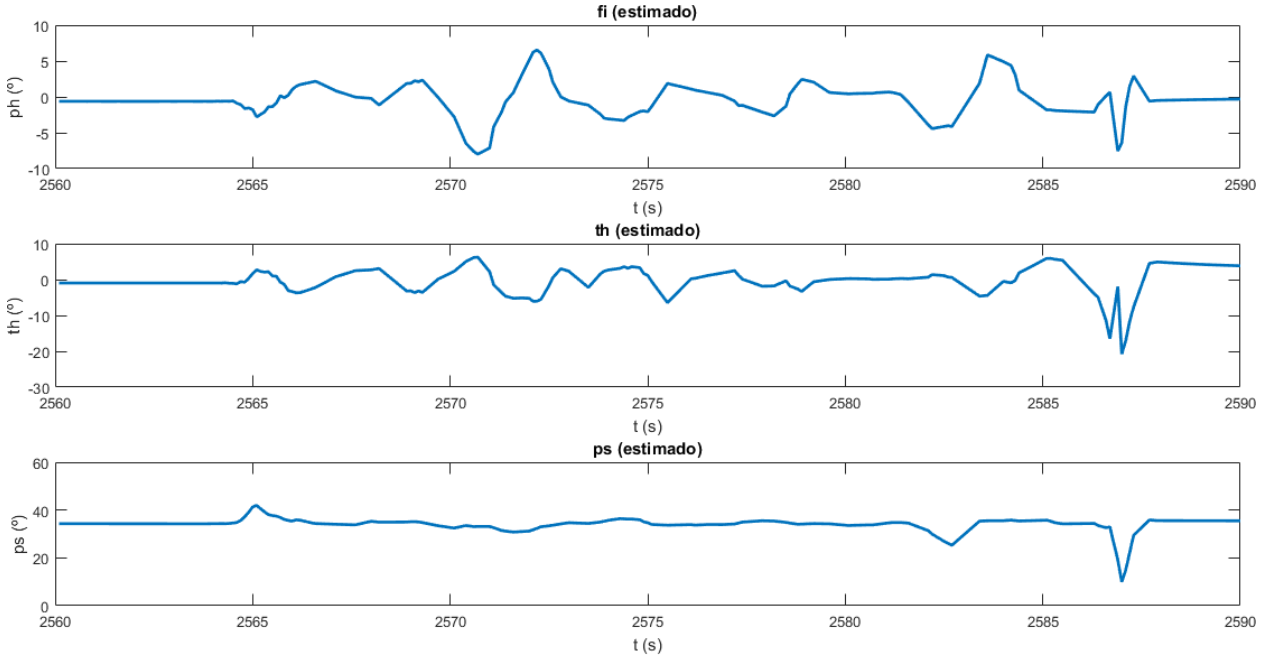


Figura 54 : Orientación estimada, vuelo P05 (20170531T15-40-36)

Otra información útil la podemos obtener si presentamos la información correspondiente a la variación temporal de los ángulos de Euler, tal como se ha hecho en la Figura 55. Se observan picos simultáneos de los tres parámetros cerca del segundo 2587. Se trata del momento en el que el vehículo golpea el suelo. Para ver correctamente los valores de las variaciones temporales es necesario eliminar la parte de choque.

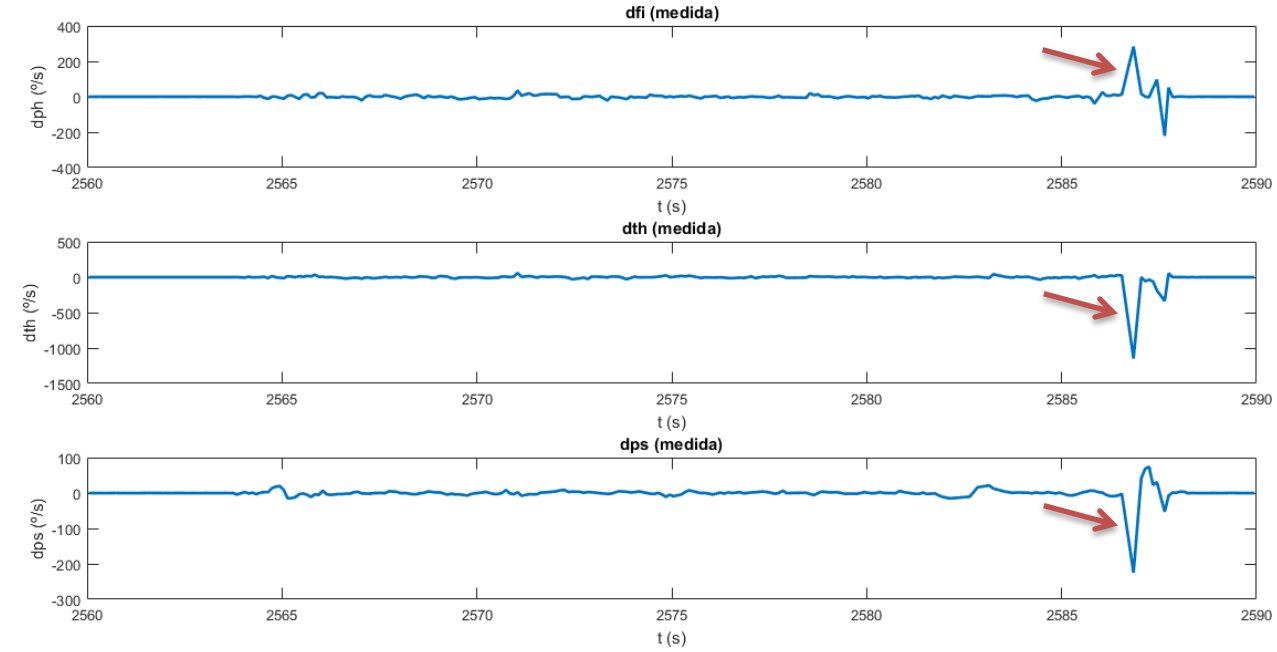


Figura 55 : Velocidad angular, vuelo P05 (20170531T15-40-36)

En la

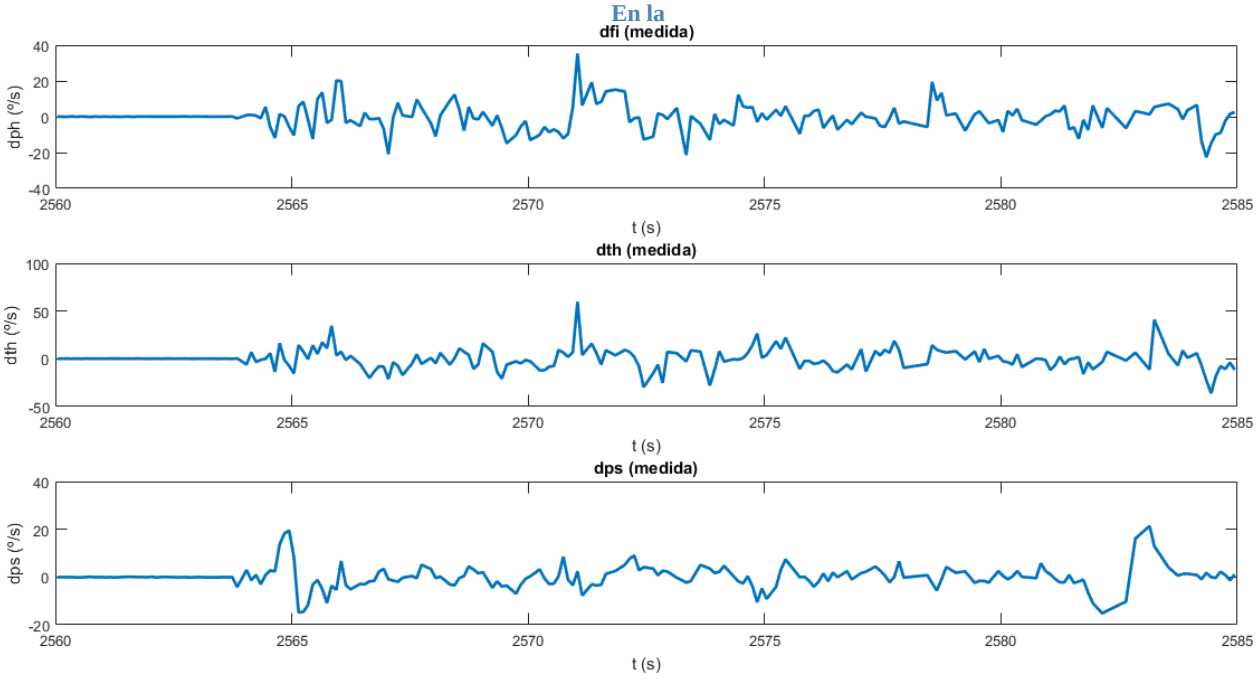


Figura 56 se han eliminado los últimos cinco segundos de datos. Ahora podemos observar el perfil de los datos de vuelo correspondientes a las velocidades angulares, que se obtiene a partir del giróscopo. Nuevamente, se trata de datos experimentales que muestran un nivel de ruido en el sensor importante, que precisan de posterior tratamiento para obtener información a partir de ellos.

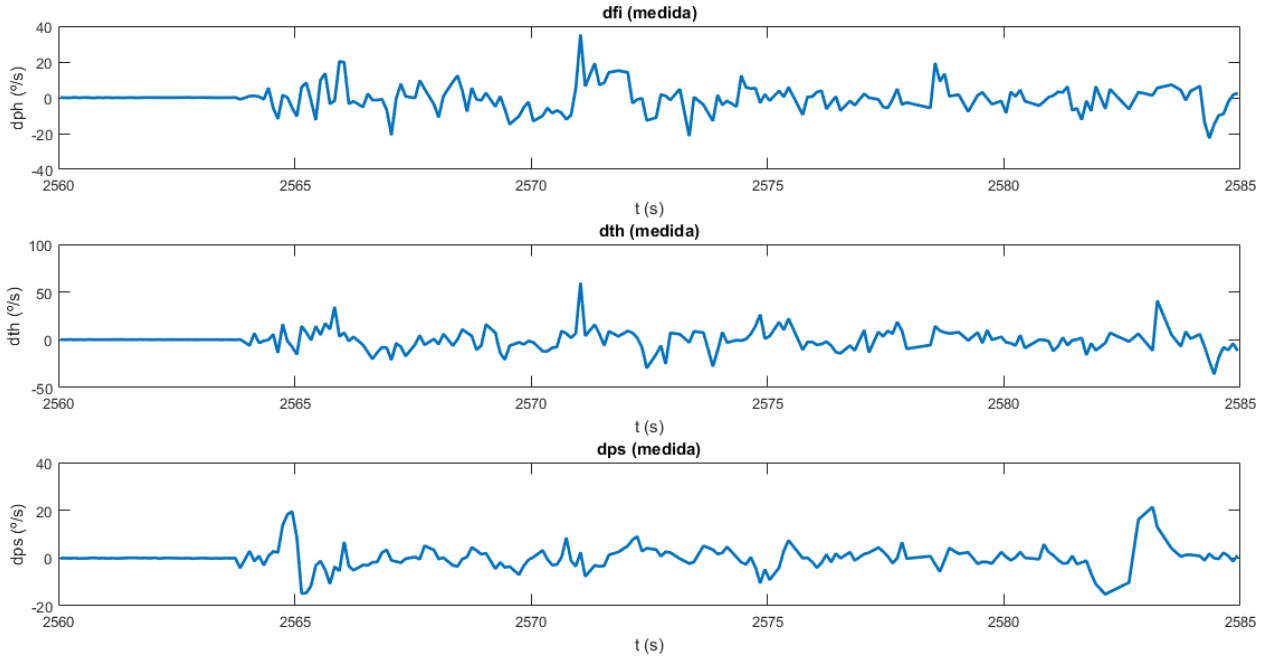


Figura 56 : Velocidad angular (sin choque), vuelo P05 (20170531T15-40-36)

Lo mismo ocurre con la medida de las aceleraciones proporcionada por el acelerómetro, que se muestra en la Figura 57. Se observa la presencia de ruido en el sensor también en este caso, y la presencia entorno al segundo 2587 de valores altos que indican el impacto con el suelo.

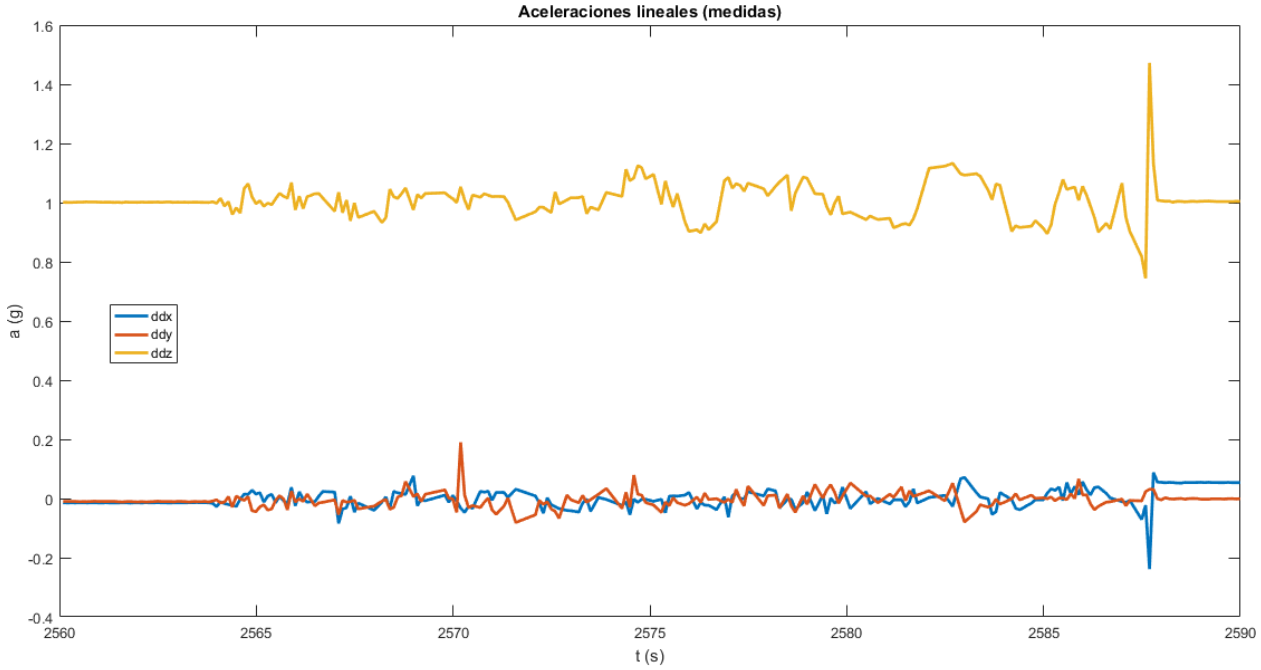


Figura 57 : Aceleración lineal, vuelo P05 (20170531T15-40-36)

Un aspecto importante de las características del vehículo que ha limitado notablemente el ejercicio de validación ha consistido en la falta de una medición adecuada de la altitud de vuelo.

En la Figura 58 se muestran las distintas posibilidades disponibles a través de la telemetría para dicha magnitud.

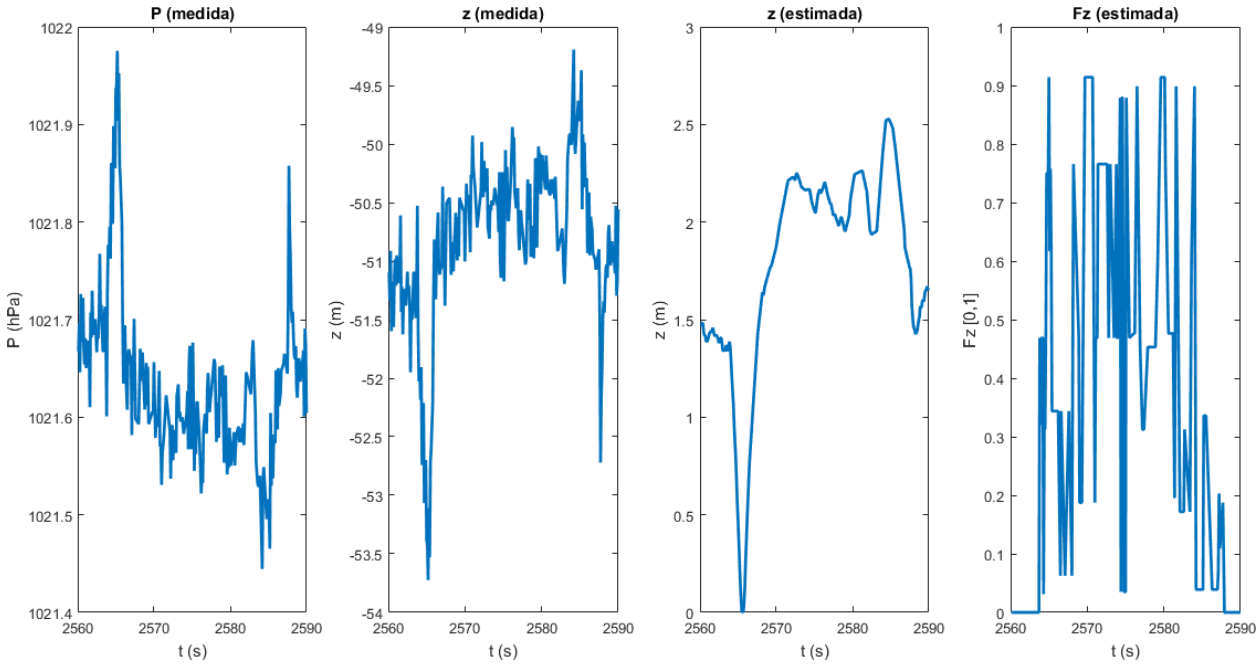


Figura 58 : Medición de la altitud, vuelo P05 (20170531T15-40-36)

Tal y como se mencionó en el apartado 4.2.2, incluyendo un cálculo teórico y una prueba preliminar para más detalle, el sensor barométrico del vehículo tiene una exactitud de ±10 Pa, que se traduce en aproximadamente ±80 cm en la estimación de la altitud de vuelo, que es claramente insuficiente para los vuelos de prueba realizados y que plantea problemas para poder simular correctamente dichos vuelos.

En la Figura 58 podemos observar las mediciones y estimaciones que obtenemos del vehículo. La primera es la medición de presión del barómetro, donde aparece el ruido del sensor, pero que muestra un claro perfil de presiones a lo largo del vuelo. Dicho perfil es el mismo que se presenta en la segunda gráfica, invertido en este caso dado que ahora se trata de medida de altitud. Como el cliente de la estación de tierra no cuenta con la posibilidad de recoger una medida de la presión estática del nivel de vuelo cero en el entorno de los ensayos para realizar los cálculos de altitud correctamente, los niveles de vuelo mostrados son en referencia a la presión al nivel del mar de la atmósfera standard internacional [25] introducida en los cálculos del firmware. Como consecuencia de ello, los valores de altitud mostrados no son reales ya que no cuentan con la referencia de presión necesaria. No obstante podemos observar que el perfil de altitud medido se puede corresponder aproximadamente con el del vuelo real, aunque presenta un claro punto de lectura incorrecta (outliner) en el momento del despegue. En la tercera gráfica se muestra la estimación de altitud realizada por el firmware y en la que se ha introducido un desplazamiento a cero del valor menor. Nuevamente la presencia del outliner y de la falta de exactitud en la lectura hace que la estimación de altitud tenga una falta de precisión importante. Comparando el perfil de altitud estimado con los valores estimados de tracción, presentados en la cuarta gráfica y normalizados para el intervalo [0,1], y con el video del vuelo grabado, parece claro que la estimación de altitud no se puede usar para la validación sin realizar un tratamiento apropiado de dicha señal, que va más allá de las actividades del presente trabajo.

Como consecuencia de lo anterior, el punto de inyección de señal situado en el bloque que simula la estación de tierra, y que es el punto necesario para poder validar correctamente el simulador desarrollado, no se puede usar en este trabajo, ya que dado que no se dispone de una estimación adecuada de altitud, y ninguna estimación de posición horizontal por falta de sensores en el vehículo, no se cuenta con las señales apropiadas para ser inyectadas. Queda como actividad futura la posibilidad de realizar una validación adecuada del simulador cuando se disponga de una celda de ensayos con posibilidad de captura de movimiento. Para esa futura actividad se han instalado las provisiones necesarias en el simulador.

Otro de los puntos de inyección de señal se sitúa en la entrada de la planta, tal y como se muestra en la Figura 59. Tiene por objeto suministrar al simulador la señal de voltaje con PWM suministrado a los actuadores, tal y como es medido por el firmware.

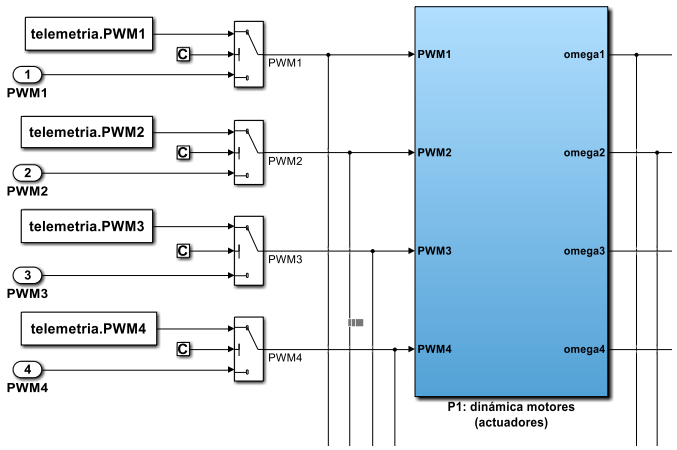


Figura 59 : Punto de inyección señal PWM

Debido al ruido presente en la señal de PWM, tal y como se ha mostrado en la Figura 52, el resultado de la mencionada inyección muestra una reproducción correcta del perfil de la fuerza aerodinámica vertical generada por los actuadores, pero no se consigue reproducir correctamente el orden de magnitud y el perfil del momento aerodinámico generado por los mismos. Esto se puede atribuir, en parte, a los bajos valores de los momentos que, tal y como vimos en los resultados de la verificación reproducidos en el ANEXO 4, tienen un orden de magnitud habitualmente de 10^{-5} N·m o menor, por lo que el ruido en el sensor se propaga al inyectar la señal en el simulador a la simulación del momento aerodinámico. En la Figura 60 se realiza este ejercicio, tras la correspondiente configuración del simulador para el tramo del vuelo P05 tratado hasta ahora. Se presenta la comparación entre los valores de fuerza y momento aerodinámico generado por el simulador (en azul) y los valores deseados de dichos parámetros obtenidos por telemetría de los generados por el controlador del firmware (en rojo). Se puede observar la correcta reproducción de la fuerza aerodinámica vertical por el simulador mientras que el momento aerodinámico no es reproducido correctamente.

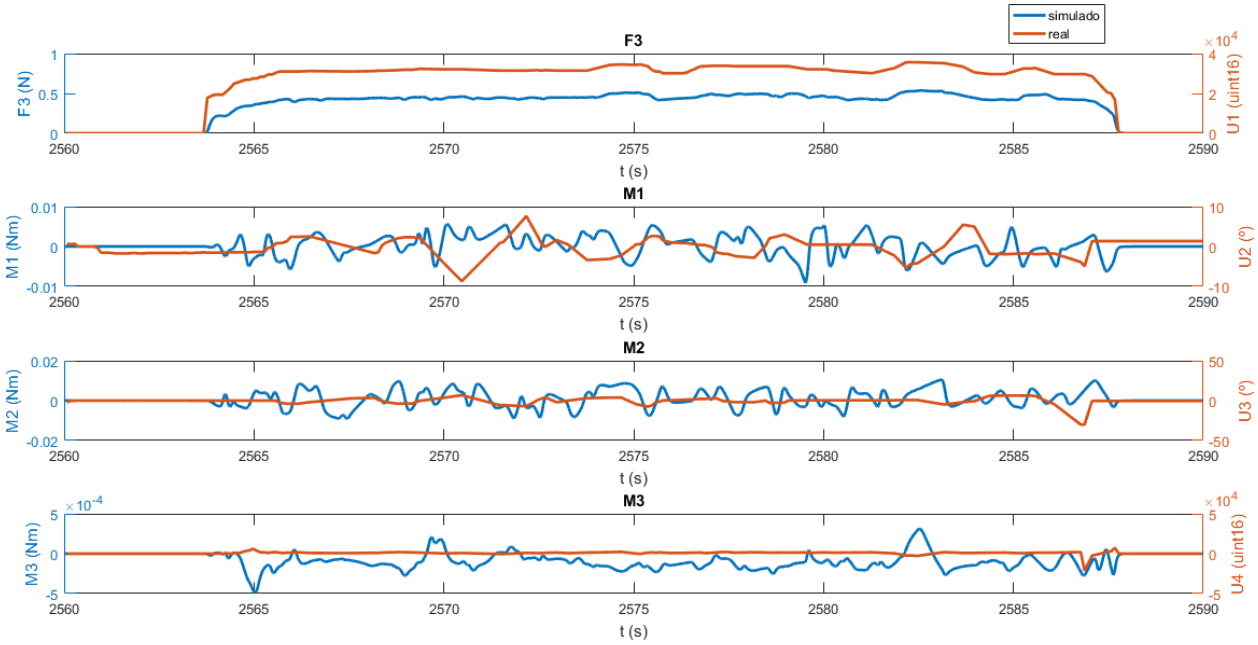


Figura 60 : Fuerza y momento aplicados, vuelo P05 (20170531T15-40-36)

La última prueba de validación a realizar consiste en emplear el tercer punto de inyección introducido en el simulador, que se mostró en la Figura 49. Inyectando la señal en ese punto se cubre tanto la simulación de la planta propulsora como la simulación del vehículo. Tras configurar el simulador convenientemente podemos visualizar los valores del voltaje vertido por el controlador a los actuadores y compararlo con el real que se ha medido en el vuelo. Dicha comparación se muestra en la Figura 61. Podemos observar que el simulador genera una señal de voltaje con un perfil suave (en azul), pero que es representativa de la señal realmente enviada a los actuadores, una vez eliminado el ruido del sensor (en rojo).

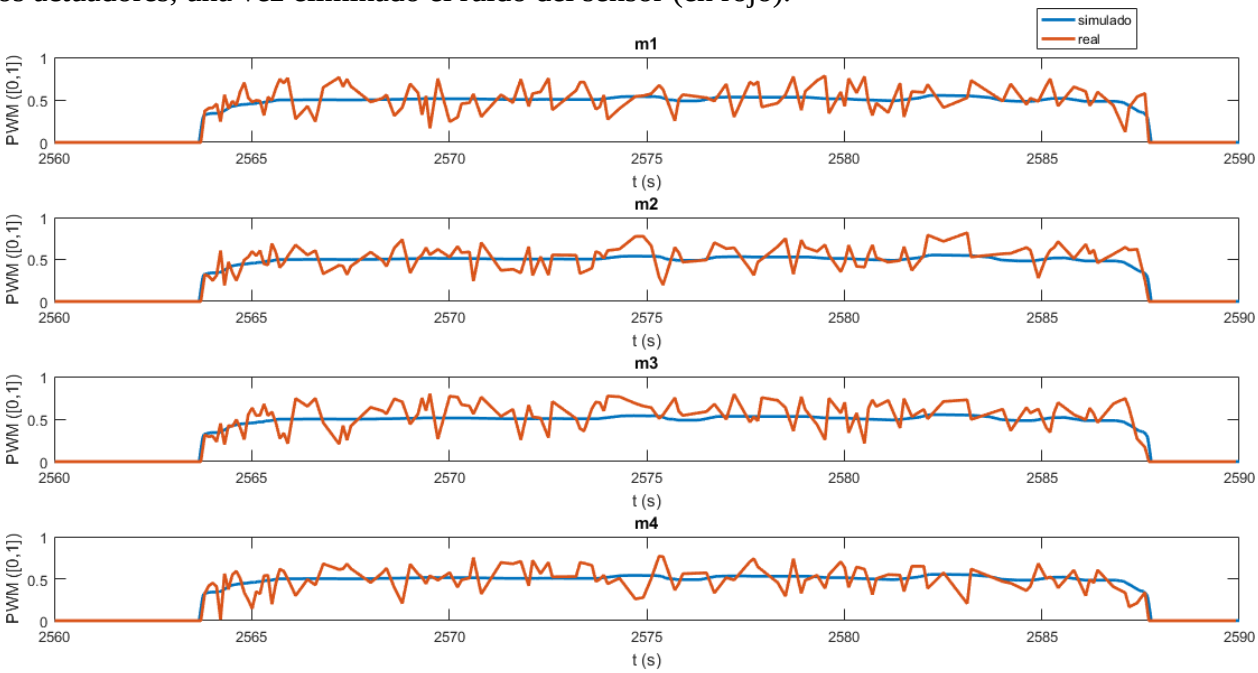


Figura 61 : PWM simulado y aplicado, vuelo P05 (20170531T15-40-36)

También podemos comparar en este punto la fuerza y momento aerodinámicos generados por el simulador con los deseados que se han introducido. En la Figura 62 se realiza la citada comparación.

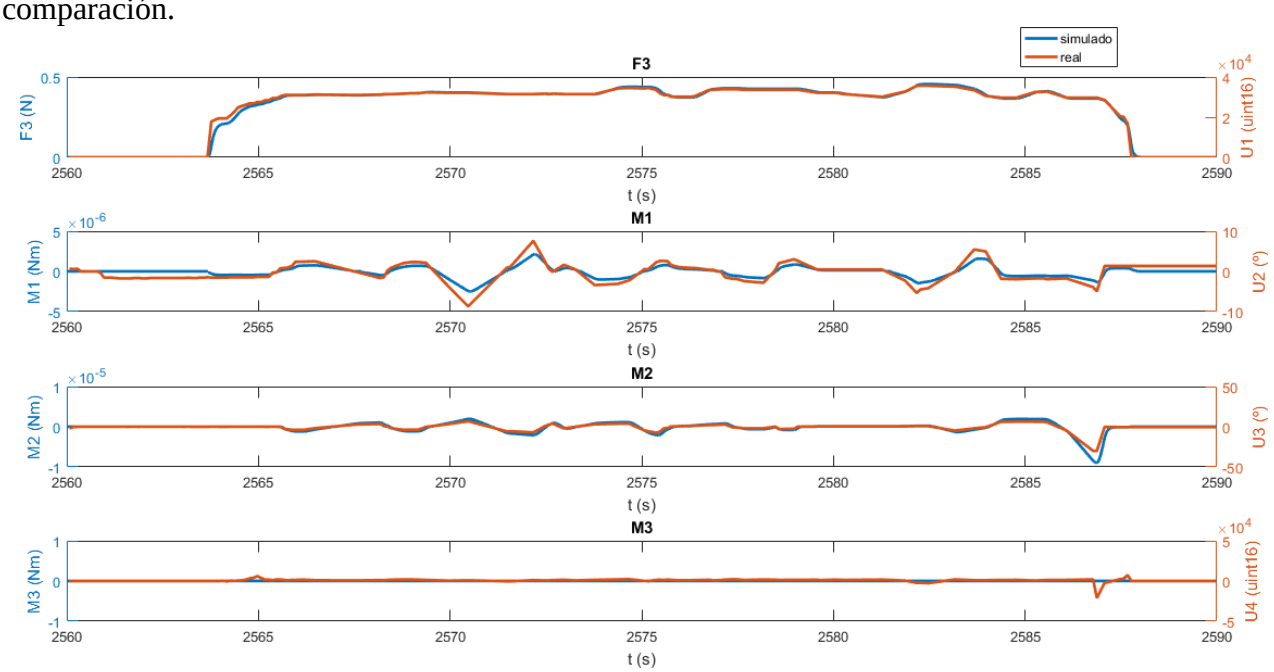


Figura 62 : Fuerza y momento simulados, vuelo P05 (20170531T15-40-36)

Se puede observar en la Figura 62 que los valores de fuerza y momento aerodinámico simulado (en azul) son muy próximos a los obtenidos por telemetría (en rojo) y que los órdenes de magnitud de dichas variables son también congruentes con los obtenidos durante la verificación.

Realizando simulaciones parecidas con otros tramos de vuelo correspondientes al resto de vuelos de prueba se obtienen resultados parecidos, por lo que podemos concluir que el simulador puede considerarse parcialmente validado. Esto es así porque en las pruebas que hemos realizado se han inyectado señales de poca calidad y en puntos internos al lazo de control, mientras que para una validación completa se precisaría inyectar la señal de referencia de entrada de la estación de tierra, es decir la señal de posición deseada y ángulo de guiñada deseado con un grado de calidad aceptable. Queda para un futuro desarrollo conseguir completar la validación con la ayuda de una celda de captura de movimiento tipo MOCAP o uno de los nuevos sistemas de posicionamiento local por radiofrecuencia descritos en el apartado 9.1.

10 CONCLUSIONES Y TRABAJO FUTURO

El presente trabajo ha perseguido realizar la simulación mediante ordenador de un sistema físico existente, tratando de capturar sus características más relevantes. Dado que se ha tratado de un proyecto académico, el resultado final, es decir el simulador desarrollado, dista mucho de un producto comercial finalizado, debiendo considerarse más bien como un primer prototipo de desarrollo en el que aún existen muchas posibilidades de mejora.

De hecho, el principal objetivo del trabajo consistía en aplicar las competencias adquiridas a lo largo del master en un proyecto concreto, tratando de emplear las técnicas de investigación adecuadas y documentando el proceso y los resultados. En este sentido, el trabajo ha sido una fuente excepcional de aprendizaje para el autor, que ha tenido que poner en práctica la mayoría de las mencionadas competencias a lo largo del mismo y, por tanto, se puede considerar que se ha cubierto el mencionado objetivo.

Una fase importante del trabajo ha sido la de documentación bibliográfica, tanto genérica respecto al estado de conocimiento de la dinámica de los cuatrirotores, como específica respecto a las especificaciones del vehículo objeto de estudio y de su firmware. Ambas han sido necesarias, y claves en el desarrollo del proyecto dado que se optó por un modelado matemático basado en leyes físicas, que precisa de un análisis detallado previo de la realidad que queremos modelar, así como de un conocimiento profundo de las leyes que la rigen.

Los resultados han mostrado que el simulador desarrollado reproduce los aspectos principales de la dinámica del vehículo objeto de estudio, y que se puede emplear para desarrollar y realizar pruebas simuladas de nuevos controladores para dicho vehículo en el futuro.

Existen muchas tareas que continúan abiertas para la mejora del prototipo inicial del simulador. Algunas de ellas se han ido mencionando a lo largo de la memoria (mejora de la visualización, mejora de la simulación de la estación de tierra, simulación de los sensores, mejora en los módulos de simulación de entorno, etc.). Tal vez la tarea más interesante que se podría completar en el futuro sería la conexión del simulador a una arena o test bed que contara con un sistema de captura de movimiento. Dicha posibilidad permitiría perfeccionar el simulador y completar la validación del mismo, ya que se contaría con datos muy precisos de localización. Otra tarea que podría ser interesante para la mejora del simulador consiste en la integración de todas las posibilidades de configuración del firmware, ya que el controlador se ha desarrollado replicando únicamente una de las configuraciones posibles. Finalmente, también sería interesante explorar en el futuro la posibilidad de generar el código del firmware del controlador de forma automática a partir de un modelo de controlador nuevo desarrollado en el simulador. Disponer de esta posibilidad cerraría el bucle de desarrollo de un nuevo controlador con esta herramienta evitando posibles errores generados por la codificación manual del firmware.

En términos de tiempo empleado en el trabajo fin de master, la fase de desarrollo del simulador y la de elaboración de la memoria han consumido una cantidad de horas muy superior a la establecida en el programa del Master (12 créditos ECTS). Por un lado, escribir una memoria con el suficiente nivel de detalle y con referencias bibliográficas adecuadas es una labor que requiere una cantidad de tiempo importante. Por otro, el desarrollo del simulador ha consumido una cantidad ingente de tiempo. Esto ha sido así porque en primer lugar, y antes de lanzarse a la fase de desarrollo hubo que emplear muchas horas familiarizándose con gran cantidad de

información para afrontar con éxito la tarea (características del vehículo, principios físicos principales, estudios anteriores sobre el tema). Asimismo, el firmware desarrollado por el fabricante, que fue necesario estudiar, era de una complejidad importante y no estaba documentado, lo que consumió muchas horas que fueron empleadas en su estudio y en el proceso de prueba y error para su comprensión. Finalmente, la mayor cantidad de horas del proyecto se consumió en el diseño y depuración del simulador. La fase de depuración, especialmente, fue muy larga debido a pequeños errores introducidos en la fase de diseño del simulador que llevó mucho tiempo identificar.

BIBLIOGRAFÍA

[1] «Crazyflie 2.0 | Bitcraze». [En línea]. Disponible en: https://www.bitcraze.io/crazyflie-2/. [Accedido: 09-oct-2016].

[2] «Loco positioning in LTH Math Department | Bitcraze». [En línea]. Disponible en: https://www.bitcraze.io/2016/06/loco-positioning-in-lth-math-department/. [Accedido: 21-may-2017].

[3] «Distributed Flying and Localisation». [En línea]. Disponible en: http://control.ee.ethz.ch/~distfly/. [Accedido: 21-may-2017].

[4] J. Förster, «System Identification of the Crazyflie 2.0 Nano Quadrocopter», Institute for Dynamic Systems and Control Swiss Federal Institute of Technology (ETH) Zurich, 2015.

[5] «Jaskirat_Singh_UAV_TargetFollowing_Thesis_1.2.pdf - Google Drive». [En línea]. Disponible en: https://drive.google.com/file/d/0B25OtRh9Sj8PYTh6MDlFZ1o2blU/view. [Accedido: 21-may-2017].

[6] «Automatic Control Laboratory». [En línea]. Disponible en: https://control.ee.ethz.ch/index.cgi?page=sada;action=details;id=715. [Accedido: 21-may-2017].

[7] A. Ledergerber, M. Hamer, y R. D'Andrea, «A robot self-localization system using one-way ultra-wideband communication», presentado en 2015 IEEE7RSJ International conference on Intelligent Robots and Systems (IROS), 2015, pp. 3131-3137.

[8] «Robust Control of an Unmanned Aerial Vehicle with Collision Avoidance - RaM Literature Repository - Aigaion 2.0». [En línea]. Disponible en: https://www.ram.ewi.utwente.nl/aigaion/publications/show/2495. [Accedido: 21-may-2017].

[9] Benoit Landry, «Planning and Control for Quadrotor Flight through Cluttered Environments», Massachusetts Institute of Technology, Master thesis, 2015.

[10] S. Bansal, A. K. Akametalu, F. J. Jiang, F. Laine, y C. J. Tomlin, «Learning Quadrotor Dynamics Using Neural Network for Flight Control», *ArXiv161005863 Cs Math*, oct. 2016.

[11] «Luis_LeNy_nanoquad.pdf». [En línea]. Disponible en: http://www.professeurs.polymtl.ca/jerome.le-ny/docs/reports/Luis_LeNy_nanoquad.pdf. [Accedido: 21-may-2017].

[12] B. Galea, E. Kia, N. Aird, y P. G. Kry, «Stippling with Aerial Robots», en *Proceedings of the Joint Symposium on Computational Aesthetics and Sketch Based Interfaces and Modeling and Non-Photorealistic Animation and Rendering*, Aire-la-Ville, Switzerland, Switzerland, 2016, pp. 125–134.

[13] «GitHub - USC-ACTLab/crazyswarm: A Large Nano-Quadcopter Swarm». [En línea]. Disponible en: https://github.com/USC-ACTLab/crazyswarm. [Accedido: 21-may-2017].

[14] «Mixed reality for robotics - IEEE Xplore Document». [En línea]. Disponible en: http://ieeexplore.ieee.org.ezproxy.uned.es/document/7354138/. [Accedido: 21-may-2017].

[15] L. Kamrath y J. Hereford, «Development of Autonomous Quadcopter», 2017, pp. 1-6.

[16] R. C. S. Nakano, A. Bandala, G. E. Faelden, J. M. Maiiiiigo, y E. P. Dadios, «Implementation of an artificial neural network in recognizing in-flight quadrotor images», 2015, pp. 1-5.

[17] L. Campos-Macias, D. Gomez-Gutierrez, R. Aldana-Lopez, R. de la Guardia, y J. I. Parra-Vilchis, «A Hybrid Method for Online Trajectory Planning of Mobile Robots in Cluttered Environments», *IEEE Robot. Autom. Lett.*, vol. 2, n.º 2, pp. 935-942, abr. 2017.

[18] «Home | Bitcraze». [En línea]. Disponible en: https://www.bitcraze.io/. [Accedido: 08-oct-2016].

[19] «Bitcraze · GitHub». [En línea]. Disponible en: https://github.com/bitcraze. [Accedido: 22-feb-2017].

[20] «The GNU General Public License v3.0 - GNU Project - Free Software Foundation». [En línea]. Disponible en: http://www.gnu.org/copyleft/gpl.html. [Accedido: 22-feb-2017].
[21] «MPU-9250 | InvenSense». [En línea]. Disponible en: https://www.invensense.com/products/motion-tracking/9-axis/mpu-9250/. [Accedido: 09-oct-2016].
[22] «Crazyflie 2.0 - Spare 7x16 mm coreless DC motor with connector». [En línea]. Disponible en: https://www.seeedstudio.com/Crazyflie-20-Spare-7x16-mm-coreless-DC-motor-with-connector-p-2115.html?cPath=84_147. [Accedido: 09-oct-2016].
[23] «LPS25H - MEMS pressure sensor: 260-1260 hPa absolute digital output barometer - STMicroelectronics». [En línea]. Disponible en: http://www.st.com/content/st_com/en/products/mems-and-sensors/pressure-sensors/lps25h.html. [Accedido: 08-oct-2016].
[24] «Crazyflie Nano Quadcopter - 4 x CW+CCW spare propellers (BC-CWP-01-A and BC-CCWP-01-A)». [En línea]. Disponible en: https://www.seeedstudio.com/Crazyflie-Nano-Quadcopter-4-x-CWCCW-spare-propellers-BCCWP01A-and-BCCCWP01A-p-1361.html?cPath=84_114. [Accedido: 09-oct-2016].
[25] «ISO 2533:1975(en), Standard Atmosphere». [En línea]. Disponible en: https://www.iso.org/obp/ui/#iso:std:iso:2533:ed-1:v1:en. [Accedido: 22-oct-2016].
[26] «Knowledge of drive technology and motion control.» [En línea]. Disponible en: http://www.maxonmotor.com/maxon/view/content/maxon-Knowledge. [Accedido: 14-ene-2017].
[27] «STM32F405/415 - STMicroelectronics». [En línea]. Disponible en: http://www.st.com/en/microcontrollers/stm32f405-415.html?querycriteria=productId=LN1035. [Accedido: 26-ene-2017].
[28] «nRF51822 / Bluetooth low energy / Products / Home - Ultra Low Power Wireless Solutions from NORDIC SEMICONDUCTOR». [En línea]. Disponible en: https://www.nordicsemi.com/eng/Products/Bluetooth-low-energy/nRF51822. [Accedido: 05-feb-2017].
[29] «projects:crazyflie2:architecture:index [Bitcraze Wiki]». [En línea]. Disponible en: https://wiki.bitcraze.io/projects:crazyflie2:architecture:index. [Accedido: 09-oct-2016].
[30] «GitHub - bitcraze/crazyflie-firmware: The main firmware for the Crazyflie Nano Quadcopter.» [En línea]. Disponible en: https://github.com/bitcraze/crazyflie-firmware. [Accedido: 01-feb-2017].
[31] «projects:crazyflie2:hardware:schematics [Bitcraze Wiki]». [En línea]. Disponible en: https://wiki.bitcraze.io/projects:crazyflie2:hardware:schematics. [Accedido: 09-oct-2016].
[32] «GitHub - bitcraze/crazyflie-clients-python at 2016.4.1». [En línea]. Disponible en: https://github.com/bitcraze/crazyflie-clients-python/tree/2016.4.1. [Accedido: 23-feb-2017].
[33] «projects:virtualmachine:index [Bitcraze Wiki]». [En línea]. Disponible en: https://wiki.bitcraze.io/projects:virtualmachine:index. [Accedido: 23-feb-2017].
[34] «Crazyradio PA | Bitcraze». [En línea]. Disponible en: https://www.bitcraze.io/crazyradio-pa/. [Accedido: 23-feb-2017].
[35] «MATLAB - El lenguaje del cálculo técnico». [En línea]. Disponible en: https://es.mathworks.com/products/matlab.html. [Accedido: 12-feb-2017].
[36] «Simulink - Simulation and Model-Based Design». [En línea]. Disponible en: https://es.mathworks.com/products/simulink.html. [Accedido: 12-feb-2017].
[37] «Eclipse - The Eclipse Foundation open source community website.» [En línea]. Disponible en: https://eclipse.org/. [Accedido: 21-may-2017].

[38] «GCC, the GNU Compiler Collection - GNU Project - Free Software Foundation (FSF)». [En línea]. Disponible en: https://gcc.gnu.org/. [Accedido: 21-may-2017].
[39] «Seeed Studio Bazaar, Boost ideas, Extend the Reach». [En línea]. Disponible en: https://www.seeedstudio.com/?gclid=CNzjw5rEgNQCFVAQ0wod3KILNQ. [Accedido: 21-may-2017].
[40] «Crazyflies quad drone unboxing - YouTube». [En línea]. Disponible en: https://www.youtube.com/watch?v=RnDrPQCfHUc. [Accedido: 21-may-2017].
[41] «Getting started with the Crazyflie 2.0 | Bitcraze». [En línea]. Disponible en: https://www.bitcraze.io/getting-started-with-the-crazyflie-2-0/. [Accedido: 12-sep-2016].
[42] «Oracle VM VirtualBox». [En línea]. Disponible en: https://www.virtualbox.org/. [Accedido: 12-sep-2016].
[43] «GitHub - bitcraze/bitcraze-vm at 2016.06». [En línea]. Disponible en: https://github.com/bitcraze/bitcraze-vm/tree/2016.06. [Accedido: 21-may-2017].
[44] S. Bouabdallah y R. Siegwart, «Full control of a quadrotor», 2007, pp. 153-158.
[45] S. Lupashin, M. Hehn, M. W. Mueller, A. P. Schoellig, M. Sherback, y R. D'Andrea, «A platform for aerial robotics research and demonstration: The Flying Machine Arena», *Mechatronics*, vol. 24, n.º 1, pp. 41-54, feb. 2014.
[46] M. Hehn y R. D'Andrea, «Real-time trajectory generation for interception maneuvers with quadrocopters», 2012, pp. 4979-4984.
[47] M. W. Mueller, M. Hehn, y R. D'Andrea, «A Computationally Efficient Motion Primitive for Quadrocopter Trajectory Generation», *IEEE Trans. Robot.*, vol. 31, n.º 6, pp. 1294-1310, dic. 2015.
[48] S. Lupashin, A. Schöllig, M. Sherback, y R. D'Andrea, «A simple learning strategy for high-speed quadrocopter multi-flips», en *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 2010, pp. 1642–1648.
[49] M. Jun y R. D'Andrea, «Path planning for unmanned aerial vehicles in uncertain and adversarial environments», en *Cooperative control: models, applications and algorithms*, Springer US, 2003, pp. 95–110.
[50] S. Zingg, D. Scaramuzza, S. Weiss, y R. Siegwart, «MAV navigation through indoor corridors using optical flow», 2010, pp. 3361-3368.
[51] M. Neunert *et al.*, «Fast nonlinear Model Predictive Control for unified trajectory optimization and tracking», 2016, pp. 1398-1404.
[52] N. Michael, D. Mellinger, Q. Lindsey, y V. Kumar, «The GRASP Multiple Micro-UAV Testbed», *IEEE Robot. Autom. Mag.*, vol. 17, n.º 3, pp. 56-65, sep. 2010.
[53] D. Mellinger y V. Kumar, «Minimum snap trajectory generation and control for quadrotors», 2011, pp. 2520-2525.
[54] A. Concha, G. Loianno, V. Kumar, y J. Civera, «Visual-inertial direct SLAM», 2016, pp. 1331-1338.
[55] G. Loianno *et al.*, «A swarm of flying smartphones», 2016, pp. 1681-1688.
[56] M. Bangura, M. Melega, R. Naldi, y R. Mahony, «Aerodynamics of Rotor Blades for Quadrotors», *ArXiv160100733 Phys.*, ene. 2016.
[57] R. Mahony, V. Kumar, y P. Corke, «Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor», *IEEE Robot. Autom. Mag.*, vol. 19, n.º 3, pp. 20-32, sep. 2012.
[58] P. I. Corke, *Robotics, vision and control: fundamental algorithms in MATLAB*. Berlin: Springer, 2011.
[59] P. Pounds, R. Mahony, y P. Corke, «Modelling and control of a quad-rotor robot», en *Proceedings Australasian Conference on Robotics and Automation 2006*, 2006.

[60] P. Pounds, R. Mahony, y P. Corke, «Modelling and control of a large quadrotor robot», *Control Eng. Pract.*, vol. 18, n.º 7, pp. 691–699, 2010.
[61] P. Pounds, R. Mahony, P. Hynes, y J. M. Roberts, «Design of a four-rotor aerial robot», en *Proceedings of the 2002 Australasian Conference on Robotics and Automation (ACRA 2002)*, 2002, pp. 145–150.
[62] P. Martin y E. Salaun, «The true role of accelerometer feedback in quadrotor control», 2010, pp. 1623-1629.
[63] P.-J. Bristeau, P. Martin, E. Salaün, y N. Petit, «The role of propeller aerodynamics in the model of a quadrotor UAV», en *Control Conference (ECC), 2009 European*, 2009, pp. 683–688.
[64] N. Guenard, T. Hamel, y V. Moreau, «Dynamic modeling and intuitive control strategy for an "X4-flyer"», 2005, vol. 1, pp. 141-146.
[65] O. Dunkley, J. Engel, J. Sturm, y D. Cremers, «Visual-Inertial Navigation for a Camera-Equipped 25g Nano-Quadrotor», en *IROS2014 Aerial Open Source Robotics Workshop*, 2014.
[66] J. Engel, J. Sturm, y D. Cremers, «Camera-based navigation of a low-cost quadrocopter», en *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, 2012, pp. 2815–2821.
[67] B. Michini, J. Redding, N. Kemal Ure, M. Cutler, y J. P. How, «Design and flight testing of an autonomous variable-pitch quadrotor», 2011, pp. 2978-2979.
[68] G. Hoffmann, D. G. Rajnarayan, S. L. Waslander, D. Dostal, J. S. Jang, y C. J. Tomlin, «The Stanford testbed of autonomous rotorcraft for multi agent control (STARMAC)», en *Digital Avionics Systems Conference, 2004. DASC 04. The 23rd*, 2004, vol. 2, p. 12–E.
[69] Haomiao Huang, G. M. Hoffmann, S. L. Waslander, y C. J. Tomlin, «Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering», 2009, pp. 3277-3282.
[70] M. Santos, V. Lopez, y F. Morata, «Intelligent fuzzy controller of a quadrotor», 2010, pp. 141-146.
[71] E. Besada-Portas, J. A. Lopez-Orozco, J. Aranda, y J. M. de la Cruz, «Virtual and remote practices for learning control topics with a 3DOF quadrotor», *IFAC Proc. Vol.*, vol. 46, n.º 17, pp. 78-83, 2013.
[72] H. Liu, Y. Bai, G. Lu, y Y. Zhong, «Robust attitude control of uncertain quadrotors», *IET Control Theory Appl.*, vol. 7, n.º 11, pp. 1583-1589, jul. 2013.
[73] Z. Song y K. Sun, «Adaptive backstepping sliding mode control with fuzzy monitoring strategy for a kind of mechanical system», *ISA Trans.*, vol. 53, n.º 1, pp. 125-133, ene. 2014.
[74] J. de Jesus Rubio, J. Humberto Perez Cruz, Z. Zamudio, y A. J. Salinas, «Comparison of two quadrotor dynamic models», *IEEE Lat. Am. Trans.*, vol. 12, n.º 4, pp. 531-537, jun. 2014.
[75] A. Nagaty, S. Saeedi, C. Thibault, M. Seto, y H. Li, «Control and Navigation Framework for Quadrotor Helicopters», *J. Intell. Robot. Syst.*, vol. 70, n.º 1-4, pp. 1-12, abr. 2013.
[76] W. Dong, G.-Y. Gu, X. Zhu, y H. Ding, «Development of a Quadrotor Test Bed — Modelling, Parameter Identification, Controller Design and Trajectory Generation», *Int. J. Adv. Robot. Syst.*, vol. 12, n.º 2, p. 7, feb. 2015.
[77] A. A. Mian y W. Daobo, «Modeling and Backstepping-based Nonlinear Control Strategy for a 6 DOF Quadrotor Helicopter», *Chin. J. Aeronaut.*, vol. 21, n.º 3, pp. 261-268, jun. 2008.

[78] A. Tzes, G. Nikolakopoulos, y K. Alexis, «Model predictive quadrotor control: attitude, altitude and position experimental studies», *IET Control Theory Appl.*, vol. 6, n.º 12, pp. 1812-1827, ago. 2012.
[79] T. Dierks y S. Jagannathan, «Output Feedback Control of a Quadrotor UAV Using Neural Networks», *IEEE Trans. Neural Netw.*, vol. 21, n.º 1, pp. 50-66, ene. 2010.
[80] E. Bregu, N. Casamassima, D. Cantoni, L. Mottola, y K. Whitehouse, «Reactive Control of Autonomous Drones», 2016, pp. 207-219.
[81] K. Alexis, G. Nikolakopoulos, y A. Tzes, «Switching model predictive attitude control for a quadrotor helicopter subject to atmospheric disturbances», *Control Eng. Pract.*, vol. 19, n.º 10, pp. 1195-1207, oct. 2011.
[82] M. Chipofya, D. J. Lee, y K. T. Chong, «Trajectory Tracking and Stabilization of a Quadrotor Using Model Predictive Control of Laguerre Functions», *Abstr. Appl. Anal.*, vol. 2015, pp. 1-11, 2015.
[83] L. Derafa, T. Madani, y A. Benallegue, «Dynamic Modelling and Experimental Identification of Four Rotors Helicopter Parameters», en *2006 IEEE International Conference on Industrial Technology*, 2006, pp. 1834-1839.
[84] «BIPM - Resolution 10 of the 22nd CGPM». [En línea]. Disponible en: http://www.bipm.org/en/CGPM/db/22/10/. [Accedido: 25-feb-2017].
[85] S. Gupte, Paul Infant Teenu Mohandas, y J. M. Conrad, «A survey of quadrotor Unmanned Aerial Vehicles», 2012, pp. 1-6.
[86] A. Chovancová, T. Fico, Ľ. Chovanec, y P. Hubinsk, «Mathematical Modelling and Parameter Identification of Quadrotor (a survey)», *Procedia Eng.*, vol. 96, pp. 172-181, 2014.
[87] K. Ogata, *Ingeniería de control moderna*. Madrid: Pearson Educación, 2010.
[88] A. Barrientos, *Fundamentos de robótica*. Madrid: McGraw-Hill, Interamericana de España, 2007.
[89] E. E. Larrabee, «Practical Design of Minimum Induced Loss Propellers», 1979.
[90] «FreeRTOS - Market leading RTOS (Real Time Operating System) for embedded systems with Internet of Things extensions». [En línea]. Disponible en: http://www.freertos.org/. [Accedido: 01-feb-2017].
[91] «GitHub - bitcraze/crazyflie-firmware at 2016.09». [En línea]. Disponible en: https://github.com/bitcraze/crazyflie-firmware/tree/2016.09. [Accedido: 05-feb-2017].
[92] «misc:investigations:thrust [Bitcraze Wiki]». [En línea]. Disponible en: https://wiki.bitcraze.io/misc:investigations:thrust. [Accedido: 02-may-2017].
[93] «Gazebo». [En línea]. Disponible en: http://gazebosim.org/. [Accedido: 28-may-2017].
[94] «Coppelia Robotics V-REP: Create. Compose. Simulate. Any Robot.» [En línea]. Disponible en: http://www.coppeliarobotics.com/. [Accedido: 28-may-2017].
[95] «Motion Capture Systems | VICON». [En línea]. Disponible en: https://www.vicon.com/. [Accedido: 23-feb-2017].
[96] «OptiTrack - Motion Capture Systems». [En línea]. Disponible en: http://optitrack.com/. [Accedido: 01-may-2017].
[97] «Loco Positioning system | Bitcraze». [En línea]. Disponible en: https://www.bitcraze.io/loco-pos-system/. [Accedido: 01-may-2017].
[98] «misc:investigations:lps-precision [Bitcraze Wiki]». [En línea]. Disponible en: https://wiki.bitcraze.io/misc:investigations:lps-precision. [Accedido: 02-may-2017].
[99] «3D modeling for everyone | SketchUp». [En línea]. Disponible en: http://www.sketchup.com/es. [Accedido: 27-may-2017].

ANEXO 1 Deducción de las matrices R_{EB} y W_{EB}

La matriz de rotación que relaciona el sistema de referencia del laboratorio E ≡ $(O_E X_E Y_E Z_E)$ y el ligado al vehículo B ≡ $(O_B X_B Y_B Z_B)$ se obtiene a partir del producto de las rotaciones individuales respecto a los ejes correspondientes tal y como se detalla a continuación:

$$\boldsymbol{R}_{EB} = \boldsymbol{R}_z(\psi) \cdot \boldsymbol{R}_y(\theta) \cdot \boldsymbol{R}_x(\phi) =$$

$$= \begin{pmatrix} cos\psi & -sen\psi & 0 \\ sen\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} cos\theta & 0 & sen\theta \\ 0 & 1 & 0 \\ -sen\theta & 0 & cos\theta \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\phi & -sen\phi \\ 0 & sen\phi & cos\phi \end{pmatrix} =$$

$$= \begin{pmatrix} cos\psi cos\theta & cos\psi sen\theta sen\phi - sen\psi cos\phi & sen\psi sen\phi + cos\psi sen\theta cos\phi \\ sen\psi cos\theta & cos\psi cos\phi + sen\psi sen\theta sen\phi & sen\psi sen\theta cos\phi - cos\psi sen\phi \\ -sen\theta & cos\theta sen\phi & cos\theta cos\phi \end{pmatrix} = \boldsymbol{R}_{EB}$$

Para obtener la matriz de transformación que relaciona la velocidad angular del vehículo con la variación temporal de los ángulos de Euler se calcula, en primer lugar la matriz inversa a partir de las matrices de rotaciones elementales involucradas y una vez obtenida se calcula su inversa.

$$\boldsymbol{\omega}_B = \boldsymbol{W}_{BE} \cdot \dot{\boldsymbol{\Phi}}$$

$$\boldsymbol{\omega}_B = \begin{pmatrix} \dot{\phi} \\ 0 \\ 0 \end{pmatrix} + \boldsymbol{R}_x(\phi) \left[\begin{pmatrix} 0 \\ \dot{\theta} \\ 0 \end{pmatrix} + \boldsymbol{R}_y(\theta) \begin{pmatrix} 0 \\ 0 \\ \dot{\psi} \end{pmatrix} \right] =$$

$$= \begin{pmatrix} \dot{\phi} \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\phi & -sen\phi \\ 0 & sen\phi & cos\phi \end{pmatrix} \left[\begin{pmatrix} 0 \\ \dot{\theta} \\ 0 \end{pmatrix} + \begin{pmatrix} cos\theta & 0 & sen\theta \\ 0 & 1 & 0 \\ -sen\theta & 0 & cos\theta \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ \dot{\psi} \end{pmatrix} \right] =$$

$$= \begin{pmatrix} 1 & 0 & -sen\theta \\ 0 & cos\phi & sen\phi cos\theta \\ 0 & -sen\phi & cos\phi cos\theta \end{pmatrix} \begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix}$$

$$\dot{\boldsymbol{\Phi}} = \boldsymbol{W}_{BE}{}^{-1} \cdot \boldsymbol{\omega}_B = \boldsymbol{W}_{EB} \cdot \boldsymbol{\omega}_B =$$

$$= \begin{pmatrix} 1 & 0 & -sen\theta \\ 0 & cos\phi & sen\phi cos\theta \\ 0 & -sen\phi & cos\phi cos\theta \end{pmatrix}^{-1} \cdot \boldsymbol{\omega}_B = \begin{pmatrix} 1 & sen\phi tan\theta & cos\phi tan\theta \\ 0 & cos\phi & -sen\phi \\ 0 & sen\phi sec\theta & cos\phi sec\theta \end{pmatrix} \cdot \boldsymbol{\omega}_B$$

$$\boldsymbol{W}_{EB} = \begin{pmatrix} 1 & sen\phi tan\theta & cos\phi tan\theta \\ 0 & cos\phi & -sen\phi \\ 0 & sen\phi sec\theta & cos\phi sec\theta \end{pmatrix}$$

ANEXO 2 Configuración y parámetros firmware fabricante

Configuración

En el presente trabajo se ha optado por mantener una configuración fija del controlador del fabricante, con el mínimo de modos automáticos y de compensación activados, para facilitar el desarrollo y la validación de la simulación. En posteriores trabajos se puede avanzar en la simulación de dichos modos. Las variables de configuración empleadas son las que se muestran en la Tabla 24.

Tabla 24 : Configuración del firmware empleada

Variable	Valor	Comentario
altHoldMode	0 (false)	Modo que mantiene la altitud en un valor constante. Está todavía en desarrollo y debido a la poca exactitud del barómetro (ver 4.2.2), no es estable. Desactivado
posHoldMode	0 (false)	Modo que mantiene la posición en un valor constante. Precisa de señal externa de posición (ver 9.1). Desactivado
posSetMode	0 (false)	Modo que mantiene la posición de un enjambre de vehículos. Desactivado
yawMode	2 (xmode)	Define la configuración del vehículo (+: el motor 1 define la posición delantera; X: los motores 1 y 4 definen la posición delantera). La configuración del Crazyflie 2.0 es en X.
carefreeResetFront	0 (carefree)	Modo que usa el valor de ángulo de guiñada respecto a las coordenadas del sistema inercial. Facilita el pilotaje del vehículo para pilotos con poca experiencia. Activado
stabilizationModeRoll	1 (angle)	Modo de estabilización del alabeo (0: velocidad; 1: ángulo). Modo ángulo
stabilizationModePitch	1 (angle)	Modo de estabilización del cabeceo (0: velocidad; 1: ángulo). Modo ángulo
stailizationModeYaw	1 (angle)	Modo de estabilización del guiñada (0: velocidad; 1: ángulo). Modo ángulo
tiltComp	0 (false)	Compensador que mantiene la altitud en un valor constante al inclinarse el vehículo. Está todavía en desarrollo y debido a la poca exactitud del barómetro (ver 4.2.2), no es estable. Desactivado

Ref. `commander.c, controller_pid.c`

NOTA: la simulación se ha diseñado asumiendo los valores de la Tabla 24. El uso del simulador con una configuración diferente requeriría realizar modificaciones en el mismo. Por tanto, el simulador no se ha diseñado para tener en cuenta todas las posibles configuraciones.

Parámetros controlador posición

En la Tabla 25 se listan los parámetros del controlador de posición empleados.

Tabla 25 : Parámetros controlador posición

Parámetro	Valor	Comentario
rpLimit	20°	Valor máximo del ángulo de alabeo/balanceo
PID_X_KP	25	Valor K_P del controlador PID de la variable x
PID_X_KI	0.28	Valor K_I del controlador PID de la variable x
PID_X_KD	7	Valor K_D del controlador PID de la variable x
PID_Y_KP	25	Valor K_P del controlador PID de la variable y
PID_Y_KI	0.28	Valor K_I del controlador PID de la variable y
PID_Y_KD	7	Valor K_D del controlador PID de la variable y
PID_Z_KP	30000	Valor K_P del controlador PID de la variable z
PID_Z_KI	0	Valor K_I del controlador PID de la variable z
PID_Z_KD	10000	Valor K_D del controlador PID de la variable z
THRUST_BASE	36000	Valor de empuje que mantiene aproximadamente el vuelo a punto a punto fijo compensando exactamente el peso del vehículo (escala: 0-UINT16_MAX)
THRUST_UPPER_LIMIT	45000	Valor máximo del empuje. Valor de saturación (escala: 0-UINT16_MAX)

Ref. `position_controller_pid.c`

Parámetros controlador orientación

En la Tabla 26 se listan los parámetros del controlador de orientación empleados.

Tabla 26 : Parámetros controlador orientación

Parámetro	Valor	Comentario
PID_ROLL_RATE_KP	250.0	Valor K_P del controlador PID de la variable $\dot{\phi}$
PID_ROLL_RATE_KI	500.0	Valor K_I del controlador PID de la variable $\dot{\phi}$
PID_ROLL_RATE_KD	2.5	Valor K_D del controlador PID de la variable $\dot{\phi}$
PID_ROLL_RATE_ INTEGRATION_LIMIT	33.3	Valor saturación integrador del controlador PID de la variable $\dot{\phi}$
PID_PITCH_RATE_KP	250.0	Valor K_P del controlador PID de la variable $\dot{\theta}$
PID_PITCH_RATE_KI	500.0	Valor K_I del controlador PID de la variable $\dot{\theta}$
PID_PITCH_RATE_KD	2.5	Valor K_D del controlador PID de la variable $\dot{\theta}$
PID_PITCH_RATE_ INTEGRATION_LIMIT	33.3	Valor saturación integrador del controlador PID de la variable $\dot{\theta}$
PID_YAW_RATE_KP	70.0	Valor K_P del controlador PID de la variable $\dot{\psi}$
PID_YAW_RATE_KI	16.7	Valor K_I del controlador PID de la variable $\dot{\psi}$

PID_YAW_RATE_KD	0.0	Valor K_D del controlador PID de la variable $\dot{\psi}$
PID_YAW_RATE_ INTEGRATION_LIMIT	166.7	Valor saturación integrador del controlador PID de la variable $\dot{\psi}$
PID_ROLL_KP	6.0	Valor K_P del controlador PID de la variable ϕ
PID_ROLL_KI	3.0	Valor K_I del controlador PID de la variable ϕ
PID_ROLL_KD	0.0	Valor K_D del controlador PID de la variable ϕ
PID_ROLL_ INTEGRATION_LIMIT	20.0	Valor saturación integrador del controlador PID de la variable ϕ
PID_PITCH_KP	6.0	Valor K_P del controlador PID de la variable θ
PID_PITCH_KI	3.0	Valor K_I del controlador PID de la variable θ
PID_PITCH_KD	0.0	Valor K_D del controlador PID de la variable θ
PID_PITCH_ INTEGRATION_LIMIT	20.0	Valor saturación integrador del controlador PID de la variable θ
PID_YAW_KP	6.0	Valor K_P del controlador PID de la variable ψ
PID_YAW_KI	1.0	Valor K_I del controlador PID de la variable ψ
PID_YAW_KD	0.35	Valor K_D del controlador PID de la variableψ
PID_YAW_ INTEGRATION_LIMIT	360.0	Valor saturación integrador del controlador PID de la variable ψ
INT16_MAX	32767	Valor máximo de un entero de 16 bits con signo
DEFAULT_PID_ INTEGRATION_LIMIT	5000	Valor saturación integrador por defecto

Ref. `pid.h, attitude_pid_controller.c`

Parámetros controlador motores

En la Tabla 27 se listan los parámetros del controlador de los motores empleados.

Tabla 27 : Parámetros controlador motores

Parámetro	Valor	Comentario
CMD_MAX	65536	Señal de mando máxima. Corresponde a PWM=100% (UINT16_max)
Ca	$-6.24 \cdot 10^{-4}$	Constante de la ecuación de conversión de señal de mando a voltaje
Cb	$8.80 \cdot 10^{-2}$	Constante de la ecuación de conversión de señal de mando a voltaje
Vs	3.7	Voltaje nominal batería
limitUint16	65535	Valor máximo de entero de 16 bit sin signo

Ref. `motors.c`

ANEXO 3 Archivos MATLAB® del simulador

Se reproducen a continuación todos los archivos que constituyen el simulador del Crazyflie 2.0. Para su correcto funcionamiento se deben situar todos en la misma carpeta junto con los archivos de datos de vuelo (si existen) y ejecutar con MATLAB® el archivo **`CrazySimulator_v0.m`**

Listado

```
configuracion_simulador.m
corrección.m
CrazySimulator_v0.m
figuras_datos_vuelos.m
filtro.m
Modelo_v0.slx
modo_simulador_dummy.mat
parametros_controlador.m
parametros_controlador_nuevo.m
parametros_entorno.m
parametros_planta.m
postpro.m
prepro.m
recortar.m
```

`configuracion_simulador.m`

```
function configuracion = configuracion_simulador()
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    configuracion_simulador (CRAZYSIMULATOR V.0)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Función empleada para configurar el simulador
%   Alumno: Pablo Antón Jornet
%   Asignatura: TFM (Crazyflie 2.0) Curso 2016-17
%   Profesor: José Manuel Díaz Martínez
%   Máster Universitario en ingeniería de sistemas y de control
%   Universidad Nacional de Educación a Distancia
%   Programa:   MATLAB 9.1.0.441655 (R2016b) Student Version
%   Llamadas a archivos externos: ninguna
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   En este archivo se define la configuración de la simulación y se
%   establecen los datos necesarios para su correcto funcionamiento.
%   INSTRUCCIONES:
%   1) Configurar el modo del simulador rellenando el vector de
%   configuración.
%   2) Registrar el nombre del archivo de datos de vuelo, tambien para
%   el modo simulador
%   3) Seleccionar el controlador
%   4) Configurar la presencia o ausencia de entorno
%   5) Si modo simulador -> Rellenar datos referencia y tiempo
%   6) Si modo simulador -> Rellenar condiciones iniciales
%   7) Si modo simulador -> Configurar altura despegue
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%   1) Vector de configuración del simulador
%   Modo simulador:     Es el modo por defecto.
%                       No se introducen datos experimentales externos.
%                       La simulación se genera a partir de la señal
```

```
%                       de referencia configurada manualmente.
%                       Sirve tambien para verificación del simulador y
%                       diseño nuevos controladores.
%   Modo validación:    Se introducen datos experimentales externos
%                       obtenidos por telemetría y/o MOCAP.
%                       Se puede seleccionar la señal externa a introducir.
%                       Sirve para validación del simulador.
%   [1 0 0 0] = Modo simulador
%   [0 1 0 0] = Modo validación. Señal referencia xyzpsi externa. MOCAP
%   [0 0 1 0] = Modo validación. Señal control rpy+thrust externa
%   [0 0 0 1] = Modo validación. Señal control PWM externa
configuracion.simulador = [1 0 0 0];

%   2) Nombre archivo datos de vuelo
%   Archivo obtenido de prueba en vuelo y procesado con CrazyTelemetry
%   para cualquiera de los modos del simulador. En modo simulador usar
%   dummy de archivo: 'modo_simulador_dummy.mat'
configuracion.archivo_telemetria = 'modo_simulador_dummy.mat';

%   3) Tipo de controlador a usar
%   tipo_controlador = 1 simulación controlador del fabricante
%   tipo_controlador = 0 controlador creado en Simulink
configuracion.tipo_controlador = 1;

%   4) Impacto con entorno
%   impacto_entorno = 1 se tienen en cuenta los impactos con entorno
%   impacto entorno = 0 no existe el entorno
configuracion.impacto_entorno = 1;

%   5) Modo simulador (localización deseada y tiempo solicitud)
configuracion.duracion= 60;     % Duración simulación (s)
configuracion.xd = 1;           % Localización x deseada(m)
configuracion.txd = 20;         % Tiempo entrada señal ref x (s)
configuracion.yd = 1;           % Localización y deseada(m)
configuracion.tyd = 30;         % Tiempo entrada señal ref y (s)
configuracion.zd = 1;           % Localización z deseada(m)
configuracion.tzd = 10;         % Tiempo entrada señal ref z (s)
configuracion.psid = 45;       % Localización deseada(°)
configuracion.tpsid = 40;       % Tiempo entrada señal ref psi (s)

%   6) Modo simulador (condiciones iniciales)
%   Carga de condiciones iniciales
%   Se trata del valor del vector de estado y de las revoluciones de los
%   motores al comienzo de la simulación
%   Los valores son usados como condiciones iniciales en los cuatro
%   integradores del modelo de aeronave, en la función de transferencia
%   de los motores y en los PID del controlador
%   X0 = [x0 y0 z0 fi0 th0 ps0 dx0 dy0 dz0 dfi0 dth0 dps0 ddfi0 ddth0
%         ddps0]'
%   w0 = [omega10 omega20 omega30 omega40]'
configuracion.X0 = zeros (15,1); % No cambiar (condiciones nulas)
configuracion.w0 = zeros (4,1); % No cambiar (condiciones nulas)
%   Si alguna de las condiciones iniciales no son nulas introducirla a
%   continuación según los modelos (tantas como necesites)
%   configuracion.X0(tu estado) = tu valor inicial;
%   configuracion.w0 (tu motor) = tus revoluciones iniciales;
%   Recuerda unidades: X0 = m,rad,m/s, rad/s, rad/s2
%   Recuerda unidades: w0 = rad/s
```

```
%   7) Altura despegue
%   despegue_suelo = 1 se fuerza Z0 = coordenada z suelo (entorno)
%   despegue_suelo = 0 se mantiene Z0 establecido en 6)
configuracion.despegue_suelo = 0;
end
```

correccion.m

```
function [valor] =correccion (serie)
%##########################################################################
%                    correccion (CRAZYSIMULATOR V.0)
%##########################################################################
%   Función que corrige el ángulo de guiñada para que siempre se encuentre
%   en el intervalo (-180°,+180°)
%   Alumno: Pablo Antón Jornet
%   Asignatura: TFM (Crazyflie 2.0) Curso 2016-17
%   Profesor: José Manuel Díaz Martínez
%   Máster Universitario en ingeniería de sistemas y de control
%   Universidad Nacional de Educación a Distancia
%   Programa:   MATLAB 9.1.0.441655 (R2016b) Student Version
%   Llamadas a archivos externos: ninguna
%##########################################################################

[filas,columnas] = size(serie);
for i=1:filas
for j=1:columnas
    if serie(i,j)>pi
        serie(i,j)= serie(i,j)-(2*pi);
    end
    if serie(i,j)<-pi
        serie(i,j)= serie(i,j)+(2*pi);
    end
end
end
valor=serie;
end
```

CrazySimulator_v0.m

```
%##########################################################################9####
%                        CRAZYSIMULATOR V.0
%##########################################################################
%   Simulador de la dinámica del microdron Crazyflie 2.0
%   Alumno: Pablo Antón Jornet
%   Asignatura: TFM (Crazyflie 2.0) Curso 2016-17
%   Profesor: José Manuel Díaz Martínez
%   Máster Universitario en ingeniería de sistemas y de control
%   Universidad Nacional de Educación a Distancia
%   Programa:   MATLAB 9.1.0.441655 (R2016b) Student Version
%               Simulink 8.8 (R2016b) Student Version
%   Llamadas a archivos externos: prepro.m, Modelo_v0, postpro.m
%   configuracion_simulador.m, parametros_planta.m, parametros_entorno.m,
%   parametros_controlador.m, filtro.m,correccion.m, recortar.m,
%   modo_simulador_dummy.mat,figuras_datos_vuelo.m, correccion.m
%   yyyymmddThh-mm-ss.mat (opcional),
%   parametros_controlador_nuevo.m (opcional)
%##########################################################################

%##########################################################################
%                        INICIALIZACIÓN
```

```
clc; clear all; close all
%##########################################################################
%                          PREPROCESADO
run ('prepro.m')
%##########################################################################
%                          SIMULACIÓN
open_system('Modelo_v0');
sim('Modelo_v0');
%##########################################################################
%                          POSTPROCESADO
run ('postpro.m')
%##########################################################################
```

figuras_datos_vuelos.m

```
function []= figuras_datos_vuelo (telemetria)
%##########################################################################
%                    figuras_datos_vuelo (CRAZYSIMULATOR V.0)
%##########################################################################
%   Función que genera las figuras de los datos de vuelo correspondientes
%   a un vuelo de prueba
%   Alumno: Pablo Antón Jornet
%   Asignatura: TFM (Crazyflie 2.0) Curso 2016-17
%   Profesor: José Manuel Díaz Martínez
%   Máster Universitario en ingeniería de sistemas y de control
%   Universidad Nacional de Educación a Distancia
%   Programa:   MATLAB 9.1.0.441655 (R2016b) Student Version
%   Llamadas a archivos externos: ninguna
%##########################################################################

%   Asignación nombres

%   Parámetros 01-04:
rolld = telemetria.REF.ctrltarget.roll;
pitchd = telemetria.REF.ctrltarget.pitch;
yawd = telemetria.REF.ctrltarget.yaw;
canfly = telemetria.REF.sys.canfly;

%   Parámetros 05-10:
accx = telemetria.SEN1.acc.x;
accy = telemetria.SEN1.acc.y;
accz = telemetria.SEN1.acc.z;
asl = telemetria.SEN1.baro.asl;
temp = telemetria.SEN1.baro.temp;
press = telemetria.SEN1.baro.pressure;

%   Parámetros 11-16:
droll=telemetria.SEN2.gyro.x;
dpitch=telemetria.SEN2.gyro.y;
dyaw=telemetria.SEN2.gyro.z;
magx=telemetria.SEN2.mag.x;
magy=telemetria.SEN2.mag.y;
magz=telemetria.SEN2.mag.z;

%   Parámetros 17-20:
thrust=(telemetria.STA.stabilizer.thrust)./65535;
roll=telemetria.STA.stabilizer.roll;
pitch=telemetria.STA.stabilizer.pitch;
```

```
yaw=telemetria.STA.stabilizer.yaw;

%   Parámetros 21-27:
U1=telemetria.LOC1.controller.actuatorThrust;
U2=telemetria.LOC1.controller.roll;
U3=telemetria.LOC1.controller.pitch;
U4=telemetria.LOC1.controller.yaw;
ctr_yaw=(telemetria.LOC1.controller.ctr_yaw)/65535;
z=telemetria.LOC1.posEstimatorAlt.estimatedZ;
dz=telemetria.LOC1.posEstimatorAlt.velocityZ;

%   Parámetros 34-37:
M1=telemetria.MOT1.motor.m1;
M2=telemetria.MOT1.motor.m2;
M3=telemetria.MOT1.motor.m3;
M4=telemetria.MOT1.motor.m4;

%   Parámetros 38-41:
PWM1=telemetria.MOT2.pwm.m1_pwm;
PWM2=telemetria.MOT2.pwm.m2_pwm;
PWM3=telemetria.MOT2.pwm.m3_pwm;
PWM4=telemetria.MOT2.pwm.m4_pwm;


%   Lanzar figuras
%   Fig 2 Planta propulsora
    figure('Name','Planta propulsora')
    subplot(4,1,1)
    plot (PWM1)
    title ('m1 (medido)')
    ylabel('PWM1 ([0,1])')
    xlabel ('t (s)')
    subplot(4,1,2)
    plot (PWM2)
    title ('m2 (medido)')
    ylabel('PWM2 ([0,1])')
    xlabel ('t (s)')
    subplot(4,1,3)
    plot (PWM3)
    title ('m3 (medido)')
    ylabel('PWM3 ([0,1])')
    xlabel ('t (s)')
    subplot(4,1,4)
    plot (PWM4)
    title ('m4 (medido)')
    ylabel('PWM4 ([0,1])')
    xlabel ('t (s)')

%   Fig 3 Control orientación
    figure('Name','Control orientación')
    subplot(4,1,1)
    yyaxis left
    plot (rolld)
    ylabel('fi* (°)')
    hold on
    yyaxis right
    plot (U2)
    ylabel ('U2 (°)')
    yyaxis left
```

```
    title ('fi (señal control)')
    xlabel ('t (s)')
    subplot(4,1,2)
    yyaxis left
    plot (pitchd)
    ylabel('th* (°)')
    hold on
    yyaxis right
    plot (U3)
    ylabel ('U3 (°)')
    yyaxis left
    title ('th (señal control)')
    xlabel ('t (s)')
    subplot(4,1,3)
    yyaxis left
    plot (yawd)
    ylabel('ps* (°)')
    hold on
    yyaxis right
    plot (ctr_yaw)
    ylabel ('U4 (°)')
    yyaxis left
    title ('ps (señal control)')
    xlabel ('t (s)')

%   Fig 4 Orientación
    figure('Name','Orientación')
    subplot(3,1,1)
    plot (roll)
    title ('fi (estimado)')
    ylabel('ph (°)')
    xlabel ('t (s)')
    subplot(3,1,2)
    plot (pitch)
    title ('th (estimado)')
    ylabel('th (°)')
    xlabel ('t (s)')
    subplot(3,1,3)
    plot (yaw)
    title ('ps (estimado)')
    ylabel('ps (°)')
    xlabel ('t (s)')

%   Fig 5 Velocidad angular
    figure('Name','Velociad angular')
    subplot(3,1,1)
    plot (droll)
    title ('dfi (medida)')
    ylabel('dph (°/s)')
    xlabel ('t (s)')
    subplot(3,1,2)
    plot (dpitch)
    title ('dth (medida)')
    ylabel('dth (°/s)')
    xlabel ('t (s)')
    subplot(3,1,3)
    plot (dyaw)
    title ('dps (medida)')
    ylabel('dps (°/s)')
```

```
    xlabel ('t (s)')

%   Fig 6 Aceleración lineal
    figure('Name','Aceleración lineal')
    plot (accx)
    hold on
    plot (accy)
    hold on
    plot (accz)
    ylabel('a (g)')
    legend ('ddx','ddy','ddz')
    title ('Aceleraciones lineales (medidas)')
    xlabel ('t (s)')

%   Fig 7 Eje z
    figure('Name','Eje z')
    subplot (1,4,1)
    plot (press)
    title ('P (medida)')
    ylabel('P (hPa)')
    xlabel ('t (s)')
    subplot (1,4,2)
    plot (asl)
    title ('z (medida)')
    ylabel('z (m)')
    xlabel ('t (s)')
    subplot (1,4,3)
    plot (z)
    title ('z (estimada)')
    ylabel('z (m)')
    xlabel ('t (s)')
    subplot(1,4,4)
    plot (thrust)
    title ('Fz (estimada)')
    ylabel('Fz [0,1]')
    xlabel ('t (s)')
end
```

filtro.m

```
function [valor] = filtro (entrada,error,choque,ejez)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                     filtro (CRAZYSIMULATOR V.0)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Función que elimina valores muy pequeños de señales y anula los
%   puntos de choque con el entorno
%   Alumno: Pablo Antón Jornet
%   Asignatura: TFM (Crazyflie 2.0) Curso 2016-17
%   Profesor: José Manuel Díaz Martínez
%   Máster Universitario en ingeniería de sistemas y de control
%   Universidad Nacional de Educación a Distancia
%   Programa:   MATLAB 9.1.0.441655 (R2016b) Student Version
%   Llamadas a archivos externos: ninguna
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
epsilon = error;
[filas,columnas] = size(entrada);
for i=1:filas
for j=1:columnas
    if abs(entrada(i,j))<epsilon
```

```
        entrada(i,j)=0;
    end
    if (ejez && choque (3,2,i)==1)
            entrada(i,j)=NaN;
    end
    if (choque(1,1,i)==1 || choque (1,2,i)==1)
        entrada(i,j)=NaN;
    end
    if (choque(2,1,i)==1 || choque (2,2,i)==1)
        entrada(i,j)=NaN;
    end
end
end
valor=entrada;
end
```

Modelo_v0.slx

Se trata de un modelo de Simulink® que contiene la simulación desarrollada

modo_simulador_dummy.mat

Se trata de un archivo de datos que no se usan pero es necesario para usar el modo simulación

parametros_controlador.m

```
function parametroc = parametros_controlador()
%#########################################################################
%                    parametros_controlador (CRAZYSIMULATOR V.0)
%#########################################################################
%   Función que crea una estructura con los parametros empleados en el
%   controlador tipo 0 (firmware desarrollado por fabricante)
%   Alumno: Pablo Antón Jornet
%   Asignatura: TFM (Crazyflie 2.0) Curso 2016-17
%   Profesor: José Manuel Díaz Martínez
%   Máster Universitario en ingeniería de sistemas y de control
%   Universidad Nacional de Educación a Distancia
%   Programa:   MATLAB 9.1.0.441655 (R2016b) Student Version
%   Llamadas a archivos externos: ninguna
%   NOTA:parametros empleados por el firmware del controlador STM32F405
%   Valores extraídos del código desarrollado por la empresa Bitcrazy. El
%   código está disponible en https://github.com/bitcraze
%   Versión firmware: 2016.09
%#########################################################################

%   CONVERSIÓN rad->deg
parametroc.conversion = 57.2958;

%   Parámetros controlador posición
%   Versión firmware: 2016.09
%   Archivo: position_controller_pid.c
%   Líneas código: 61-96 + 141-142

parametroc.rpLimit = 20;
parametroc.DT = 0.01;

parametroc.PID_X_KP = 25;
parametroc.PID_X_KI = 0;
parametroc.PID_X_KD = 90;
```

```
parametroc.PID_Y_KP = 25;
parametroc.PID_Y_KI = 0;
parametroc.PID_Y_KD = 90;

parametroc.PID_Z_KP = 30000;
parametroc.PID_Z_KI = 20000;
parametroc.PID_Z_KD = 10000;

parametroc.THRUST_BASE = 36000;
parametroc.THRUST_UPPER_LIMIT = 45000;
%**************************************************************************
%   Parámetros controlador orientación
%   Versión firmware: 2016.09
%   Archivo: pid.h
%   Líneas código: 34-65

parametroc.PID_ROLL_RATE_KP = 250.0;
parametroc.PID_ROLL_RATE_KI = 500.0;
parametroc.PID_ROLL_RATE_KD = 2.5;
parametroc.PID_ROLL_RATE_INTEGRATION_LIMIT = 33.3;

parametroc.PID_PITCH_RATE_KP = 250.0;
parametroc.PID_PITCH_RATE_KI = 500.0;
parametroc.PID_PITCH_RATE_KD = 2.5;
parametroc.PID_PITCH_RATE_INTEGRATION_LIMIT = 33.3;

parametroc.PID_YAW_RATE_KP = 70.0;
parametroc.PID_YAW_RATE_KI = 16.7;
%parametroc.PID_YAW_RATE_KI = 0;
parametroc.PID_YAW_RATE_KD = 0.9;
parametroc.PID_YAW_RATE_INTEGRATION_LIMIT = 166.7;

parametroc.PID_ROLL_KP = 6.0;
parametroc.PID_ROLL_KI = 3.0;
parametroc.PID_ROLL_KD = 0.0;
parametroc.PID_ROLL_INTEGRATION_LIMIT = 20.0;

parametroc.PID_PITCH_KP = 6.0;
parametroc.PID_PITCH_KI = 3.0;
parametroc.PID_PITCH_KD = 0.0;
parametroc.PID_PITCH_INTEGRATION_LIMIT = 20.0;

parametroc.PID_YAW_KP = 6.0;
parametroc.PID_YAW_KI = 1;
parametroc.PID_YAW_KD = 5;
parametroc.PID_YAW_INTEGRATION_LIMIT = 360.0;
parametroc.INT16_MAX = 32767;

parametroc.DEFAULT_PID_INTEGRATION_LIMIT = 5000.0;
end
```

parametros_controlador_nuevo.m

```
function parametroc = parametros_controlador_nuevo()
%##########################################################################
%                    parametros_controlador_nuevo (CRAZYSIMULATOR V.0)
%##########################################################################
```

```
%   Función que crea una estructura con los parametros empleados en el
%   controlador tipo 1 (nuevo diseño)
%   Alumno: Pablo Antón Jornet
%   Asignatura: TFM (Crazyflie 2.0) Curso 2016-17
%   Profesor: José Manuel Díaz Martínez
%   Máster Universitario en ingeniería de sistemas y de control
%   Universidad Nacional de Educación a Distancia
%   Programa:   MATLAB 9.1.0.441655 (R2016b) Student Version
%   Llamadas a archivos externos: ninguna
%##########################################################################

%   ESCRIBIR AQUÍ
end
```

parametros_entorno.m

```
function parametroe = parametros_entorno()
%##########################################################################
%                    parametros_entorno (CRAZYSIMULATOR V.0)
%##########################################################################
%   Función que crea una estructura con los parametros empleados en el
%   simulador para establecer las barreras físicas del entorno de vuelo
%   Alumno: Pablo Antón Jornet
%   Asignatura: TFM (Crazyflie 2.0) Curso 2016-17
%   Profesor: José Manuel Díaz Martínez
%   Máster Universitario en ingeniería de sistemas y de control
%   Universidad Nacional de Educación a Distancia
%   Programa:   MATLAB 9.1.0.441655 (R2016b) Student Version
%   Llamadas a archivos externos: ninguna
%##########################################################################

%   Barreras físicas:
%   Nota: si se quiere simular sin existencia de alguna barrera física
%   introducir 'inf' en el correspondiente parametro
%   NOTA: el parámetro x1,y1,z1 siempre tiene que ser menor que x2,y2,z2
%   Eje XE
parametroe.x1 = -inf;  % Coordenada x pared 1[m]
parametroe.x2 = inf;  % Coordenada x pared 2[m]

%   Eje YE
parametroe.y1 = -inf;  % Coordenada y pared 1[m]
parametroe.y2 = +inf;  % Coordenada y pared 2[m]
%   Eje ZE
parametroe.z1 = 0;  % Coordenada z suelo[m]
parametroe.z2 = +inf;  % Coordenada z techo[m]
end
```

parametros_planta.m

```
function parametro = parametros_planta()
%##########################################################################
%                    parametros planta (CRAZYSIMULATOR V.0)
%##########################################################################
%   Función que crea una estructura con los parametros que va a emplear
%   el modelo de la planta
%   Alumno: Pablo Antón Jornet
%   Asignatura: TFM (Crazyflie 2.0) Curso 2016-17
%   Profesor: José Manuel Díaz Martínez
```

```
%   Máster Universitario en ingeniería de sistemas y de control
%   Universidad Nacional de Educación a Distancia
%   Programa:   MATLAB 9.1.0.441655 (R2016b) Student Version
%   Llamadas a archivos externos: ninguna
%   Ref. J. Förster, «System Identification of the Crazyflie 2.0 Nano
%   Quadrocopter», Institute for Dynamic Systems and Control Swiss Federal
%   Institute of Technology (ETH) Zurich, 2015
%##########################################################################

%   PARÁMETROS PLANTA:
%   Parametros mecánicos vehículo
parametro.d = 3.97e-2;  % [m]
parametro.m = 2.7e-2; % [kg]
parametro.I = [1.66e-5 0 0; 0 1.66e-5 0; 0 0 2.93e-5];% [kg m^2]
parametro.g = 9.81; % [m s^-2]

%   Parametros aerodinamicos hélices
parametro.ct1 = 1.28e-8; % [N s^2]
parametro.ct0 = 1.56e-5; % [N s]
parametro.cm1 = 7.65e-11; % [N m s^2]
parametro.cm0 = 9.28e-8; % [N m s]


%   Parametros electricos motores
parametro.Km = 2.37e-6; % [V^-1]
parametro.Tm = 6.46e-2; % [s]
parametro.cmd = 65536;  % ithrust correspondiente a 100% PWM
parametro.ome = 28436; % relación omega


%   Parametros derivados
parametro.gama1 = 1.0e-07 .* [  +0.1280    0.1280    0.1280    0.1280;
                                -0.0051   -0.0051    0.0051    0.0051;
                                -0.0051    0.0051    0.0051   -0.0051;
                                -0.0008    0.0008   -0.0008    0.0008];
parametro.gama0 = 1.0e-04 .* [  +0.1560    0.1560    0.1560    0.1560;
                                -0.0062   -0.0062    0.0062    0.0062;
                                -0.0062    0.0062    0.0062   -0.0062;
                                -0.0009    0.0009   -0.0009    0.0009];
end
```

postpro.m

```
%##########################################################################
%                   Postprocesador (CRAZYSIMULATOR V.0)
%##########################################################################
%   Codigo para el postprocesado de los datos generados por la simulación
%   y su visualización
%   Alumno: Pablo Antón Jornet
%   Asignatura: TFM (Crazyflie 2.0) Curso 2016-17
%   Profesor: José Manuel Díaz Martínez
%   Máster Universitario en ingeniería de sistemas y de control
%   Universidad Nacional de Educación a Distancia
%   Programa:   MATLAB 9.1.0.441655 (R2016b) Student Version
%   Llamadas a archivos externos: filtro.m, correccion.m,
%   figuras_datos_vuelo.m
%##########################################################################
%   En este archivo se procesan los datos generados por el simulador
```

```
%   y se genera su visualización.
%##########################################################################

%   Conversión a grados y corrección referencia guiñada
vector_estado.fi_th_ps.Data = (vector_estado.fi_th_ps.Data).*(180/pi);
vector_estado.dfi_dth_dps.Data =(vector_estado.dfi_dth_dps.Data).*(180/pi);
referencia.psiref.Data = correccion(referencia.psiref.Data).*(180/pi);
telemetria.LOC1.posEstimatorAlt.estimatedZ=...
    (telemetria.LOC1.posEstimatorAlt.estimatedZ)-...
    min(telemetria.LOC1.posEstimatorAlt.estimatedZ);
%   Precisión presentación
epsilon.p = 1e-3;   % precisión posición
epsilon.a = 1e-3;  % presición orientación
epsilon.m = 1e-6;   % precisión resto

%   Variable auxiliar choque entorno
no_choque = zeros(size(testigo_choque.Data));

%   Filtrado vector estado y vector planta propulsora
out.x = filtro(vector_estado.x_y_z.Data(:,1),epsilon.p,...
    testigo_choque.Data,0);
out.y = filtro(vector_estado.x_y_z.Data(:,2),epsilon.p,...
    testigo_choque.Data,0);
out.z = filtro(vector_estado.x_y_z.Data(:,3),epsilon.p,...
    testigo_choque.Data,1);
out.fi = filtro(vector_estado.fi_th_ps.Data(:,1),epsilon.a,no_choque,0);
out.th = filtro(vector_estado.fi_th_ps.Data(:,2),epsilon.a,no_choque,0);
out.ps = filtro(vector_estado.fi_th_ps.Data(:,3),epsilon.a,no_choque,0);
out.dx = filtro(vector_estado.dx_dy_dz.Data(:,1),epsilon.p,no_choque,0);
out.dy = filtro(vector_estado.dx_dy_dz.Data(:,2),epsilon.p,no_choque,0);
out.dz = filtro(vector_estado.dx_dy_dz.Data(:,3),epsilon.p,no_choque,0);
out.dfi = filtro(vector_estado.dfi_dth_dps.Data(:,1),...
    epsilon.a,no_choque,0);
out.dth = filtro(vector_estado.dfi_dth_dps.Data(:,2),...
    epsilon.a,no_choque,0);
out.dps = filtro(vector_estado.dfi_dth_dps.Data(:,3),...
    epsilon.a,no_choque,0);
out.PWM1 = filtro (planta_propulsora.PWM1.Data(:,1),epsilon.m,no_choque,0);
out.PWM2 = filtro (planta_propulsora.PWM2.Data(:,1),epsilon.m,no_choque,0);
out.PWM3 = filtro (planta_propulsora.PWM3.Data(:,1),epsilon.m,no_choque,0);
out.PWM4 = filtro (planta_propulsora.PWM4.Data(:,1),epsilon.m,no_choque,0);
out.omega1 = filtro (planta_propulsora.omega1.Data(:,1),...
    epsilon.m,no_choque,0);
out.omega2 = filtro (planta_propulsora.omega2.Data(:,1),...
    epsilon.m,no_choque,0);
out.omega3 = filtro (planta_propulsora.omega3.Data(:,1),...
    epsilon.m,no_choque,0);
out.omega4 = filtro (planta_propulsora.omega4.Data(:,1),...
    epsilon.m,no_choque,0);
out.F3 = filtro (planta_propulsora.F3.Data(:,1),epsilon.m,no_choque,0);
out.M1 = filtro (planta_propulsora.M1.Data(:,1),epsilon.m,no_choque,0);
out.M2 = filtro (planta_propulsora.M2.Data(:,1),epsilon.m,no_choque,0);
out.M3 = filtro (planta_propulsora.M3.Data(:,1),epsilon.m,no_choque,0);
referencia.epsilon = epsilon;
referencia.no_choque = no_choque;
referencia.testigo_choque = testigo_choque;
clear epsilon; clear no_choque; clear testigo_choque;

%   FIGURAS MODO SIMULACIÓN
```

```
if configuracion.simulador(1) == 1

    %   Fig 1 Planta propulsora
    figure('Name','Planta propulsora')
    subplot(4,3,1)
    plot(tout,out.PWM1)
    ylabel('PWM1')
    subplot(4,3,2)
    plot(tout,out.omega1)
    ylabel('omega1 (rad/s)')
    subplot(4,3,3)
    plot(tout,out.F3)
    ylabel('F3 (N)')
    subplot(4,3,4)
    plot(tout,out.PWM2)
    ylabel('PMW2')
    subplot(4,3,5)
    plot(tout,out.omega2)
    ylabel('omega2 (rad/s)')
    subplot(4,3,6)
    plot(tout,out.M1)
    ylabel('M1 (Nm)')
    subplot(4,3,7)
    plot(tout,out.PWM3)
    ylabel('PWM3')
    subplot(4,3,8)
    plot(tout,out.omega3)
    ylabel('omega3 (rad/s)')
    subplot(4,3,9)
    plot(tout,out.M2)
    ylabel('M2 (Nm)')
    subplot(4,3,10)
    plot(tout,out.PWM4)
    ylabel('PWM4')
    subplot(4,3,11)
    plot(tout,out.omega4)
    ylabel('omega4 (rad/s)')
    subplot(4,3,12)
    plot(tout,out.M3)
    ylabel('M3 (Nm)')

    %   Fig2 Vector estado
    figure('Name','Vector de estado')
    subplot(6,2,1)
    plot(tout,out.x,tout,referencia.xref.Data)
    ylabel('x (m)')
    cuadrado= legend('real','deseado');
    set(cuadrado,'Position',[0.48 0.92 0.05 0.06]);
    clear cuadrado
    subplot(6,2,2)
    plot(tout,out.dx)
    ylabel('dx (m/s)')
    subplot(6,2,3)
    plot(tout,out.y,tout,referencia.yref.Data)
    ylabel('y (m)')
    subplot(6,2,4)
    plot(tout,out.dy)
    ylabel('dy (m/s)')
    subplot(6,2,5)
```

```
    plot(tout,out.z,tout,referencia.zref.Data)
    ylabel('z (m)')
    subplot(6,2,6)
    plot(tout,out.dz)
    ylabel('dz (m/s)')
    subplot(6,2,7)
    plot(tout,out.fi)
    ylabel('fi (°)')
    subplot(6,2,8)
    plot(tout,out.dfi)
    ylabel('dfi (°/s)')
    subplot(6,2,9)
    plot(tout,out.th)
    ylabel('th (°)')
    subplot(6,2,10)
    plot(tout,out.dth)
    ylabel('dth (°/s)')
    subplot(6,2,11)
    plot(tout,out.ps,tout,referencia.psiref.Data)
    ylabel('ps (°)')
    subplot(6,2,12)
    plot(tout,out.dps)
    ylabel('dps (°/s)')
end

%   FIGURAS MODO VALIDACIÓN
if configuracion.simulador(1) == 0

    %   Figuras datos vuelo (Fig 2-Fig 7)
    figuras_datos_vuelo(telemetria);

    %   Fig 8 PWM en los 4 motores
    figure('Name','PWM')
    subplot(4,1,1)
    plot(tout,out.PWM1)
    hold on
    plot(telemetria.MOT2.pwm.m1_pwm)
    ylabel('PWM ([0,1])')
    xlabel ('t (s)')
    title ('m1')
    cuadrado= legend('simulado','real');
    set(cuadrado,'Position',[0.78 0.92 0.05 0.06]);
    clear cuadrado
    subplot(4,1,2)
    plot(tout,out.PWM2)
    hold on
    plot(telemetria.MOT2.pwm.m2_pwm)
    ylabel('PWM ([0,1])')
    xlabel ('t (s)')
    title ('m2')
    subplot(4,1,3)
    plot(tout,out.PWM3)
    hold on
    plot(telemetria.MOT2.pwm.m3_pwm)
    ylabel('PWM ([0,1])')
    xlabel ('t (s)')
    title ('m3')
    subplot(4,1,4)
    plot(tout,out.PWM4)
```

```
hold on
plot(telemetria.MOT2.pwm.m4_pwm)
ylabel('PWM ([0,1])')
xlabel ('t (s)')
title ('m4')

%   Fig 9 F3M1M2M3
figure('Name','F3M1M2M3')
subplot(4,1,1)
yyaxis left
plot(tout,out.F3)
ylabel('F3 (N)')
hold on
yyaxis right
plot(telemetria.LOC1.controller.actuatorThrust)
ylabel('U1 (uint16)')
yyaxis left
title ('F3')
xlabel ('t (s)')
cuadrado= legend('simulado','real');
set(cuadrado,'Position',[0.78 0.92 0.05 0.06]);
clear cuadrado
subplot(4,1,2)
yyaxis left
plot(tout,out.M1)
ylabel('M1 (Nm)')
hold on
yyaxis right
plot(telemetria.LOC1.controller.roll)
ylabel('U2 (°)')
yyaxis left
title ('M1')
xlabel ('t (s)')
subplot(4,1,3)
yyaxis left
plot(tout,out.M2)
ylabel('M2 (Nm)')
hold on
yyaxis right
plot(telemetria.LOC1.controller.pitch)
ylabel('U3 (°)')
yyaxis left
title ('M2')
xlabel ('t (s)')
subplot(4,1,4)
yyaxis left
plot(tout,out.M3)
ylabel('M3 (Nm)')
hold on
yyaxis right
plot(telemetria.LOC1.controller.ctr_yaw)
ylabel('U4 (uint16)')
yyaxis left
title ('M3')
xlabel ('t (s)')

%   Fig 10 dfi, dth, dps
figure('Name','dfi, dth, dps')
subplot(3,1,1)
```

```
plot(tout,out.dfi)
hold on
plot(telemetria.SEN2.gyro.x)
title ('dfi')
ylabel('dfi (°/s)')
xlabel ('t (s)')
cuadro= legend('simulado','real');
set(cuadro,'Position',[0.78 0.92 0.05 0.06]);
clear cuadro
subplot(3,1,2)
plot(tout,out.dth)
hold on
plot(telemetria.SEN2.gyro.y)
title ('dth')
ylabel('dth (°/s)')
xlabel ('t (s)')
subplot(3,1,3)
plot(tout,out.dps)
hold on
plot(telemetria.SEN2.gyro.z)
title ('dps')
ylabel('dps (°/s)')
xlabel ('t (s)')

%   Fig 11 X (parte orientación)
figure('Name','Vector de estado (orientación)')
subplot(3,2,1)
plot(tout,out.fi)
hold on
plot(telemetria.STA.stabilizer.roll)
title ('fi')
ylabel('fi (°)')
xlabel ('t (s)')
cuadro= legend('simulado','real');
set(cuadro,'Position',[0.48 0.92 0.05 0.06]);
clear cuadro;
subplot(3,2,2)
plot(tout,out.dfi)
hold on
plot(telemetria.SEN2.gyro.x)
title ('dfi')
ylabel('dfi (°/s)')
xlabel ('t (s)')
subplot(3,2,3)
plot(tout,out.th)
hold on
plot(telemetria.STA.stabilizer.pitch)
title ('th')
ylabel('th (°)')
xlabel ('t (s)')
subplot(3,2,4)
plot(tout,out.dth)
hold on
plot(telemetria.SEN2.gyro.y)
title ('dth')
ylabel('dth (°/s)')
xlabel ('t (s)')
subplot(3,2,5)
plot(tout,out.ps)
```

```
    hold on
    plot(telemetria.STA.stabilizer.yaw)
    title ('ps')
    ylabel('ps (°)')
    xlabel ('t (s)')
    subplot(3,2,6)
    plot(tout,out.dps)
    hold on
    plot(telemetria.SEN2.gyro.z)
    title ('dps')
    ylabel('dps (°/s)')
    xlabel ('t (s)')

end
```

prepro.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                    Preprocesador (CRAZYSIMULATOR V.0)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Codigo para la carga de datos y el preprocesado de los mismos
%   Alumno: Pablo Antón Jornet
%   Asignatura: TFM (Crazyflie 2.0) Curso 2016-17
%   Profesor: José Manuel Díaz Martínez
%   Máster Universitario en ingeniería de sistemas y de control
%   Universidad Nacional de Educación a Distancia
%   Programa:   MATLAB 9.1.0.441655 (R2016b) Student Version
%   Llamadas a archivos externos: configuracion_simulador.m,
%   parametros_entorno.m, parametros_planta.m, parametros_controlador.m,
%   parametros_controlador_nuevo.m, recortar.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   En este archivo se carga la configuración del simulador,
%   la configuración del entorno, los parametros de la planta y los del
%   controlador. Asimismo, se procesan los datos para su uso durante
%   la simulación.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%   Carga configuración, entorno, parámetros planta, parámetros controlador
configuracion = configuracion_simulador (); % Estructura config simulador
parametroe = parametros_entorno (); % Estructura config entorno
parametro = parametros_planta (); % Estructura config planta
switch configuracion.tipo_controlador
    case 1
    %   Carga de parametros controlador tipo 1
    parametroc = parametros_controlador ();
    case 0
    %   Carga de parametros controlador tipo 0
    parametroc = parametros_controlador_nuevo ();
end

%   Carga datos telemetría
load (configuracion.archivo_telemetria);
%   Modo [0 1 0 0] No disponible sin MOCAP
telemetria.xref = telemetria.LOC2.posCtlAlt.targetX;
telemetria.yref = telemetria.LOC2.posCtlAlt.targetY;
telemetria.zref = telemetria.LOC2.posCtlAlt.targetZ;
telemetria.psiref = telemetria.REF.ctrltarget.yaw;
%   Modo [0 0 1 0]
```

```
telemetria.thrustref = telemetria.LOC1.controller.actuatorThrust;
telemetria.rollref  = telemetria.LOC1.controller.roll;
telemetria.pitchref = telemetria.LOC1.controller.pitch;
telemetria.yawref   = (telemetria.LOC1.controller.ctr_yaw)./65535;
%   Modo [0 0 0 1]
telemetria.PWM1 = telemetria.MOT2.pwm.m1_pwm;
telemetria.PWM2 = telemetria.MOT2.pwm.m2_pwm;
telemetria.PWM3 = telemetria.MOT2.pwm.m3_pwm;
telemetria.PWM4 = telemetria.MOT2.pwm.m4_pwm;

% Modo simulador
if configuracion.simulador(1) == 1
    configuracion.tiempo_inicial = 0;
    configuracion.tiempo_final = configuracion.duracion;
    configuracion.psid = configuracion.psid *(pi/180);
end

% Modo validacion
if configuracion.simulador(1) == 0
    %   Fig.1 Selección ventana temporal
    figure('Name','Selección ventana tiempo')
    subplot(4,1,1)
    plot(telemetria.STA.stabilizer.thrust)
    title ('Fz (estimada)')
    ylabel('Fz (uint16)')
    xlabel ('t (s)')
    subplot(4,1,2)
    plot(telemetria.STA.stabilizer.roll)
    title ('fi (estimado)')
    ylabel('fi (°)')
    xlabel ('t (s)')
    subplot(4,1,3)
    plot(telemetria.STA.stabilizer.pitch)
    title ('th (estimado)')
    ylabel('th (°)')
    xlabel ('t (s)')
    subplot(4,1,4)
    plot(telemetria.STA.stabilizer.yaw)
    title ('ps (estimado)')
    ylabel('ps(°)')
    xlabel ('t (s)')
    %   Pregunta tamaño ventana
    configuracion.tiempo_inicial = input('tiempo inicial?=');
    configuracion.tiempo_final = input('tiempo final?=');
    %   Recorte de datos
    telemetria = recortar (telemetria,configuracion.tiempo_inicial,...
    configuracion.tiempo_final);
    %   Condiciones iniciales
    configuracion.X0(1) = 0;
    configuracion.X0(2) = 0;
    configuracion.X0(3) = 0;
    configuracion.X0(4) = telemetria.STA.stabilizer.roll.Data(1)*(pi/180);
    configuracion.X0(5) = telemetria.STA.stabilizer.pitch.Data(1)*(pi/180);
    configuracion.X0(6) = telemetria.STA.stabilizer.yaw.Data(1)*(pi/180);
    configuracion.X0(7) = 0;
    configuracion.X0(8) = 0;
    configuracion.X0(9) = 0;
    configuracion.X0(10) = 0;
    configuracion.X0(11) = 0;
```

```
    configuracion.X0(12) = 0;
    configuracion.X0(13) = 0;
    configuracion.X0(14) = 0;
    configuracion.X0(15) = 0;
end
```

recortar.m

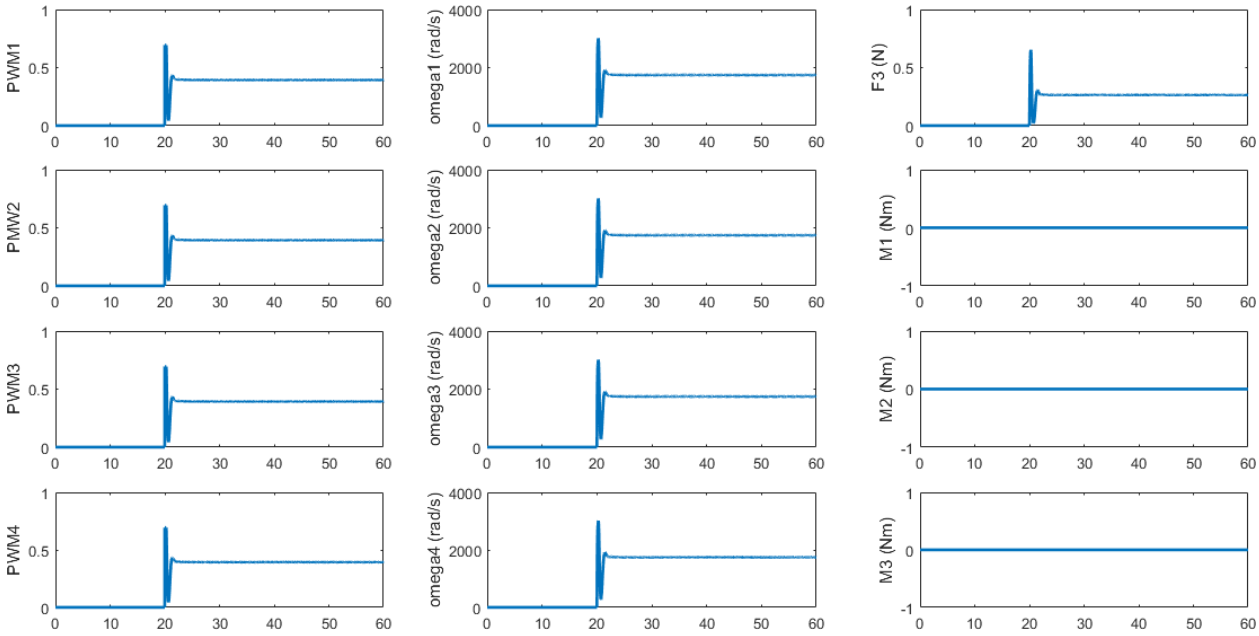
```
function telemetria=recortar(telemetria,ti,tf)
%##########################################################################
%   Función que genera los datos de vuelo obtenidos por telemetría
%   limitandolos a la ventana temporal seleccionada
%   Alumno: Pablo Antón Jornet
%   Asignatura: TFM (Crazyflie 2.0) Curso 2016-17
%   Profesor: José Manuel Díaz Martínez
%   Máster Universitario en ingeniería de sistemas y de control
%   Universidad Nacional de Educación a Distancia
%   Programa:   MATLAB 9.1.0.441655 (R2016b) Student Version
%   Llamadas a archivos externos: ninguna
%##########################################################################

telemetria.REF.ctrltarget.roll =...
    getsampleusingtime(telemetria.REF.ctrltarget.roll,ti,tf);
telemetria.REF.ctrltarget.pitch =...
    getsampleusingtime(telemetria.REF.ctrltarget.pitch,ti,tf);
telemetria.REF.ctrltarget.yaw =...
    getsampleusingtime(telemetria.REF.ctrltarget.yaw,ti,tf);
telemetria.REF.sys.canfly =...
    getsampleusingtime(telemetria.REF.sys.canfly,ti,tf);

telemetria.SEN1.acc.x =...
    getsampleusingtime(telemetria.SEN1.acc.x,ti,tf);
telemetria.SEN1.acc.y =...
    getsampleusingtime(telemetria.SEN1.acc.y,ti,tf);
telemetria.SEN1.acc.z =...
    getsampleusingtime(telemetria.SEN1.acc.z,ti,tf);
telemetria.SEN1.baro.asl =...
    getsampleusingtime(telemetria.SEN1.baro.asl,ti,tf);
telemetria.SEN1.baro.temp =...
    getsampleusingtime(telemetria.SEN1.baro.temp,ti,tf);
telemetria.SEN1.baro.pressure =...
    getsampleusingtime(telemetria.SEN1.baro.pressure,ti,tf);

telemetria.SEN2.gyro.x =...
    getsampleusingtime(telemetria.SEN2.gyro.x,ti,tf);
telemetria.SEN2.gyro.y =...
    getsampleusingtime(telemetria.SEN2.gyro.y,ti,tf);
telemetria.SEN2.gyro.z =...
    getsampleusingtime(telemetria.SEN2.gyro.z,ti,tf);
telemetria.SEN2.mag.x =...
    getsampleusingtime(telemetria.SEN2.mag.x,ti,tf);
telemetria.SEN2.mag.y =...
    getsampleusingtime(telemetria.SEN2.mag.y,ti,tf);
telemetria.SEN2.mag.z =...
    getsampleusingtime(telemetria.SEN2.mag.z,ti,tf);

telemetria.STA.stabilizer.thrust =...
    getsampleusingtime(telemetria.STA.stabilizer.thrust,ti,tf);
```

```
telemetria.STA.stabilizer.roll =...
    getsampleusingtime(telemetria.STA.stabilizer.roll,ti,tf);
telemetria.STA.stabilizer.pitch =...
    getsampleusingtime(telemetria.STA.stabilizer.pitch,ti,tf);
telemetria.STA.stabilizer.yaw =...
    getsampleusingtime(telemetria.STA.stabilizer.yaw,ti,tf);

telemetria.LOC1.controller.actuatorThrust =...
    getsampleusingtime(telemetria.LOC1.controller.actuatorThrust,ti,tf);
telemetria.LOC1.controller.roll =...
    getsampleusingtime(telemetria.LOC1.controller.roll,ti,tf);
telemetria.LOC1.controller.pitch =...
    getsampleusingtime(telemetria.LOC1.controller.pitch,ti,tf);
telemetria.LOC1.controller.yaw =...
    getsampleusingtime(telemetria.LOC1.controller.yaw,ti,tf);
telemetria.LOC1.controller.ctr_yaw =...
    getsampleusingtime(telemetria.LOC1.controller.ctr_yaw,ti,tf);
telemetria.LOC1.posEstimatorAlt.estimatedZ =...
    getsampleusingtime(telemetria.LOC1.posEstimatorAlt.estimatedZ,ti,tf);
telemetria.LOC1.posEstimatorAlt.velocityZ =...
    getsampleusingtime(telemetria.LOC1.posEstimatorAlt.velocityZ,ti,tf);

telemetria.MOT1.motor.m1 =...
    getsampleusingtime(telemetria.MOT1.motor.m1,ti,tf);
telemetria.MOT1.motor.m2 =...
    getsampleusingtime(telemetria.MOT1.motor.m2,ti,tf);
telemetria.MOT1.motor.m3 =...
    getsampleusingtime(telemetria.MOT1.motor.m3,ti,tf);
telemetria.MOT1.motor.m4 =...
    getsampleusingtime(telemetria.MOT1.motor.m4,ti,tf);

telemetria.MOT2.pwm.m1_pwm =...
    getsampleusingtime(telemetria.MOT2.pwm.m1_pwm,ti,tf);
telemetria.MOT2.pwm.m2_pwm =...
    getsampleusingtime(telemetria.MOT2.pwm.m2_pwm,ti,tf);
telemetria.MOT2.pwm.m3_pwm =...
    getsampleusingtime(telemetria.MOT2.pwm.m3_pwm,ti,tf);
telemetria.MOT2.pwm.m4_pwm =...
    getsampleusingtime(telemetria.MOT2.pwm.m4_pwm,ti,tf);
end
```

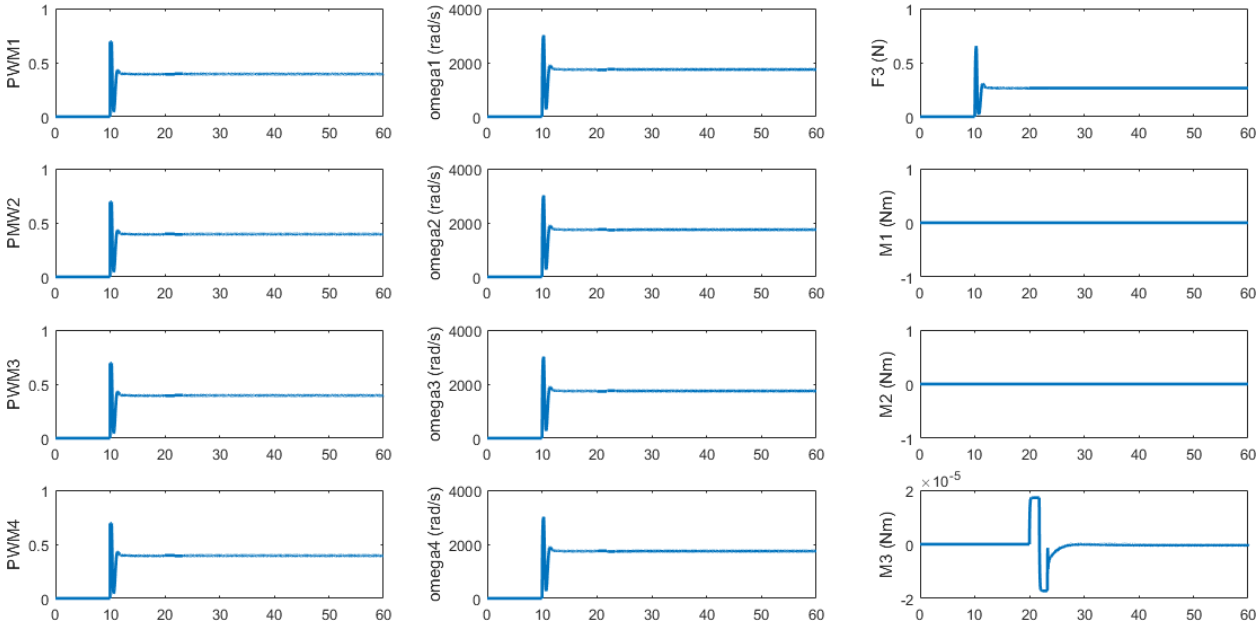
ANEXO 4 Pruebas de verificación (planta de potencia)

Se reproducen a continuación las distintas gráficas obtenidas en las pruebas de verificación para la planta de potencia. Las gráficas correspondientes al vector de estado se han reproducido en el apartado 9.2.2.

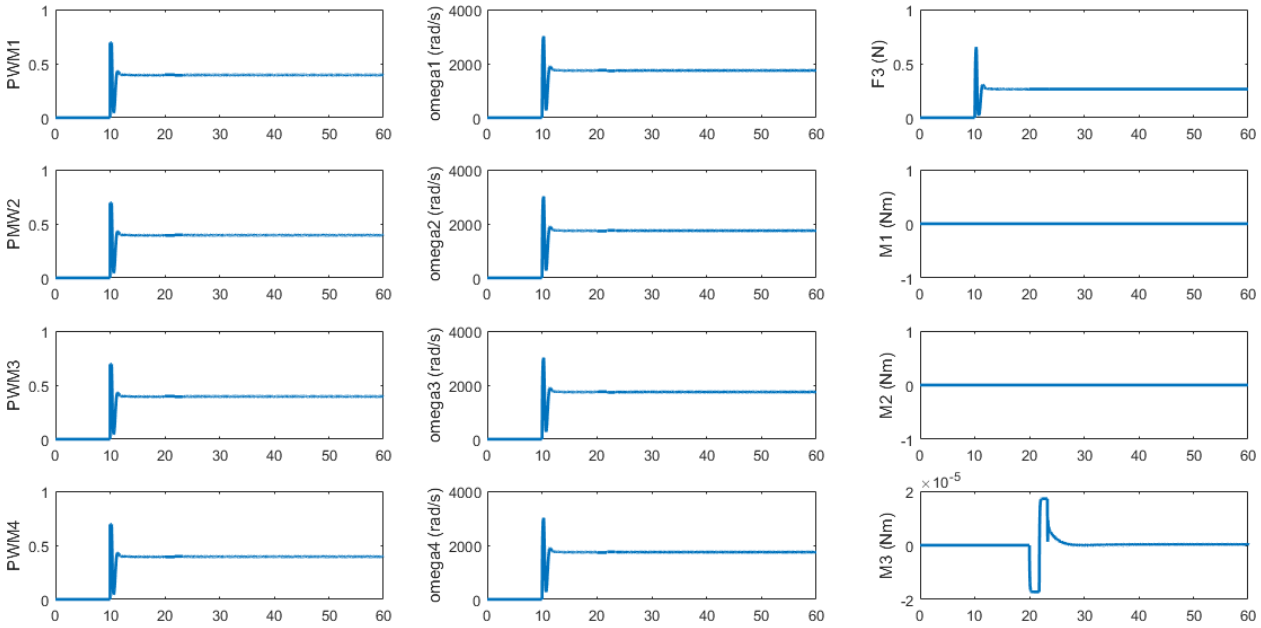
V01 (despegue y vuelo a punto fijo):



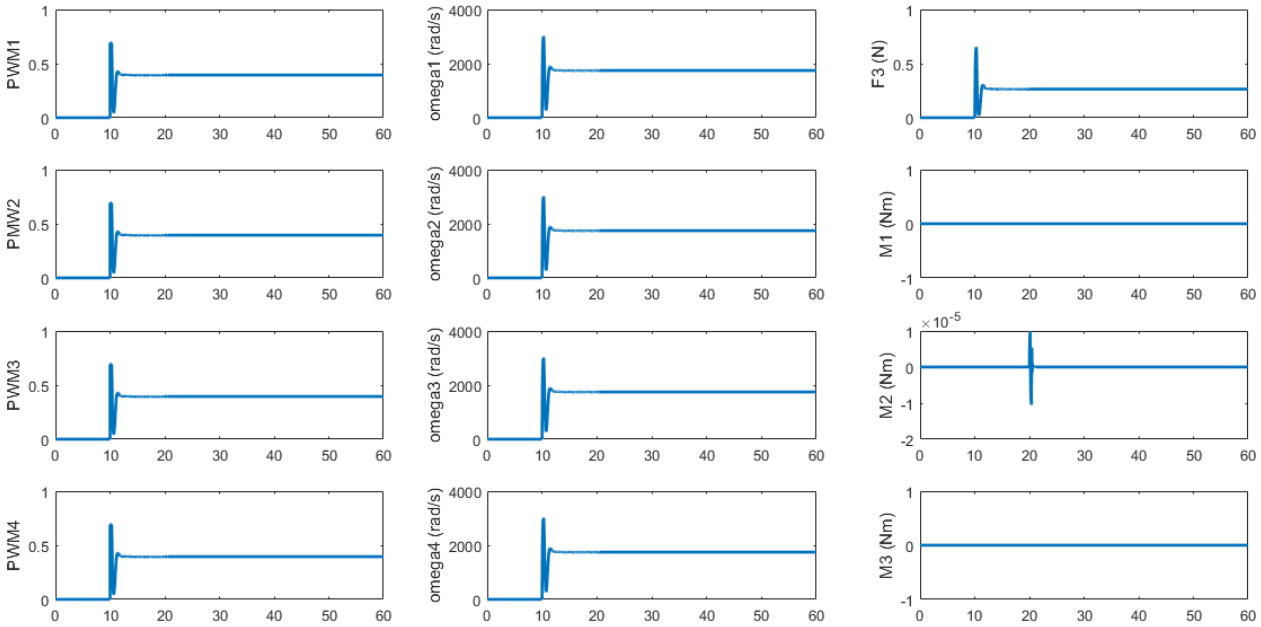
V002 (guiñada):



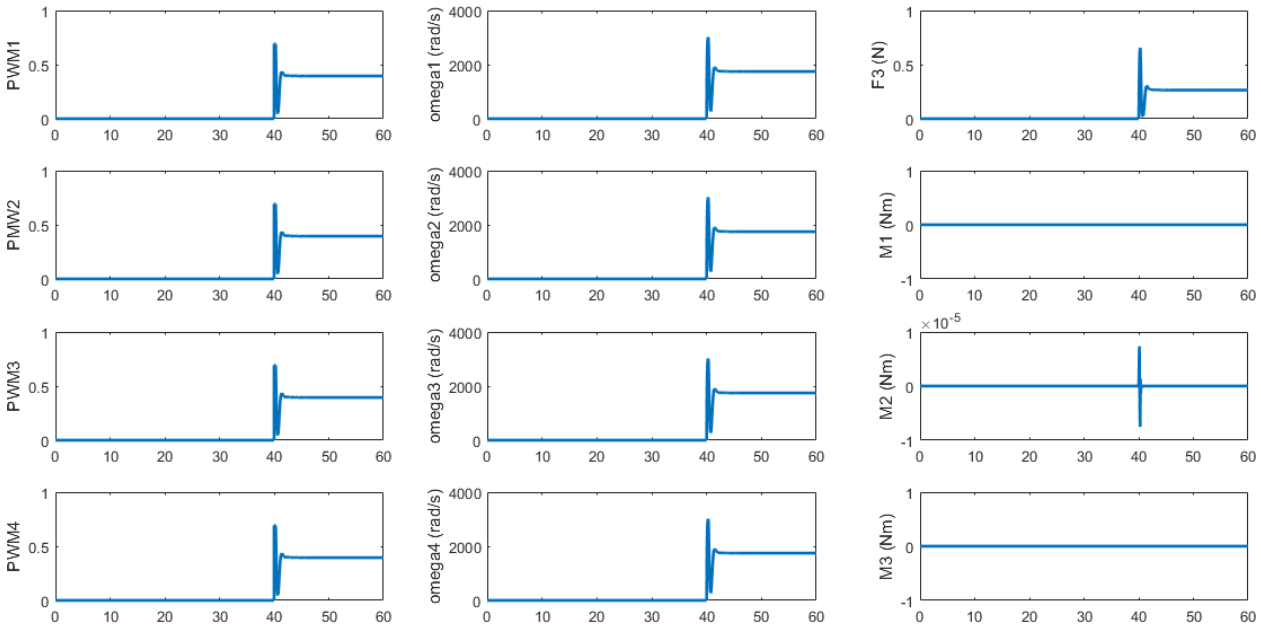
V03 (limitación ángulo guiñada):



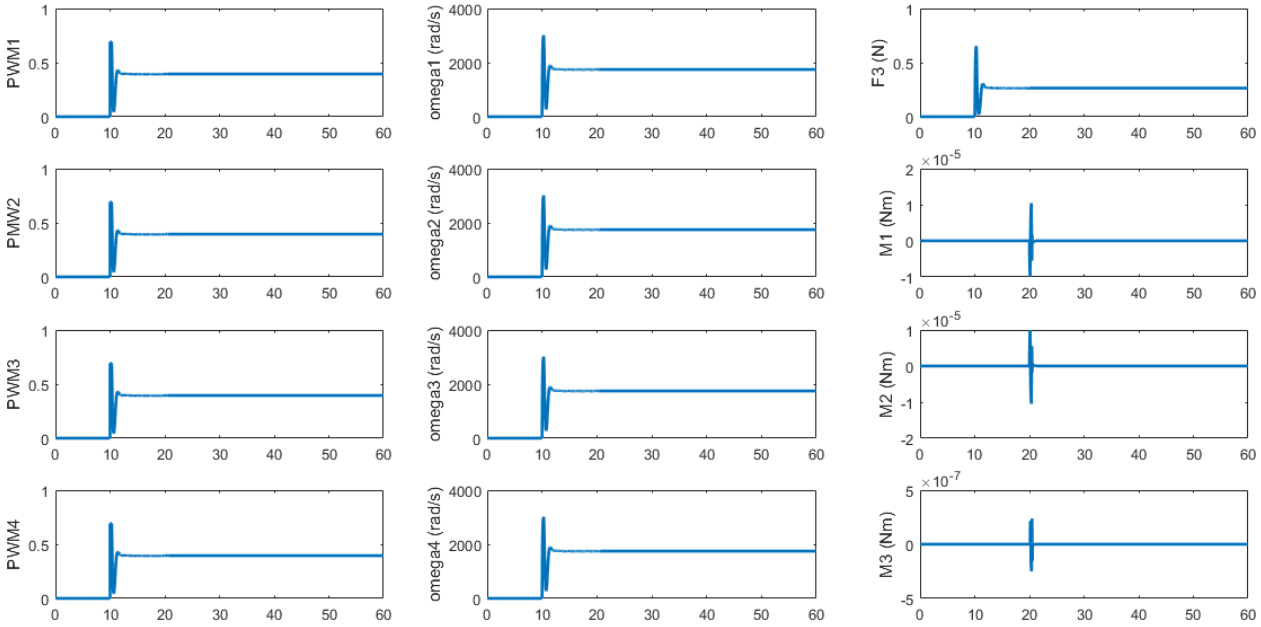
V04 (desplazamiento horizontal un eje):



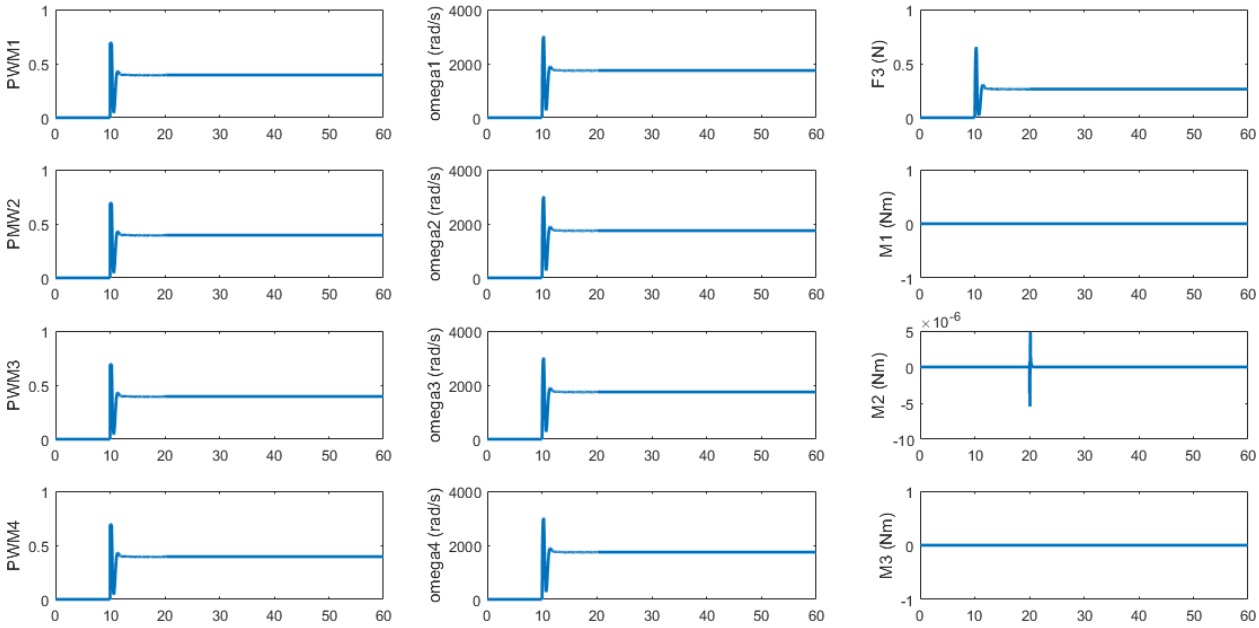
V05 (reseteo del controlador de avance):



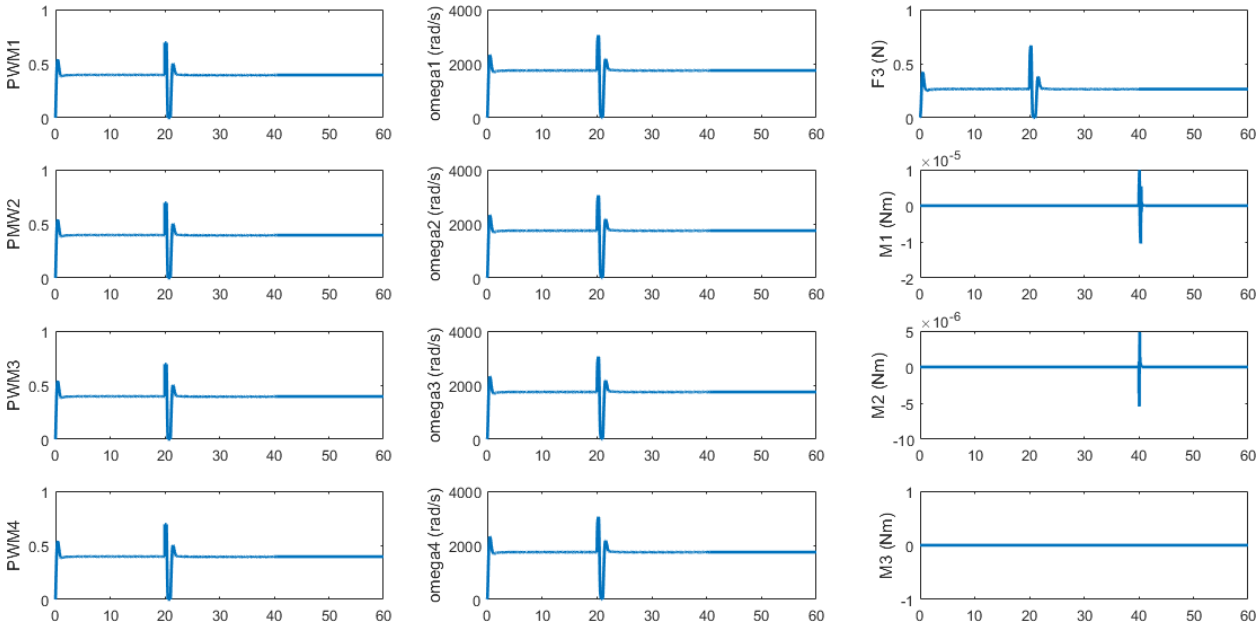
V06 (desplazamiento horizontal dos ejes):



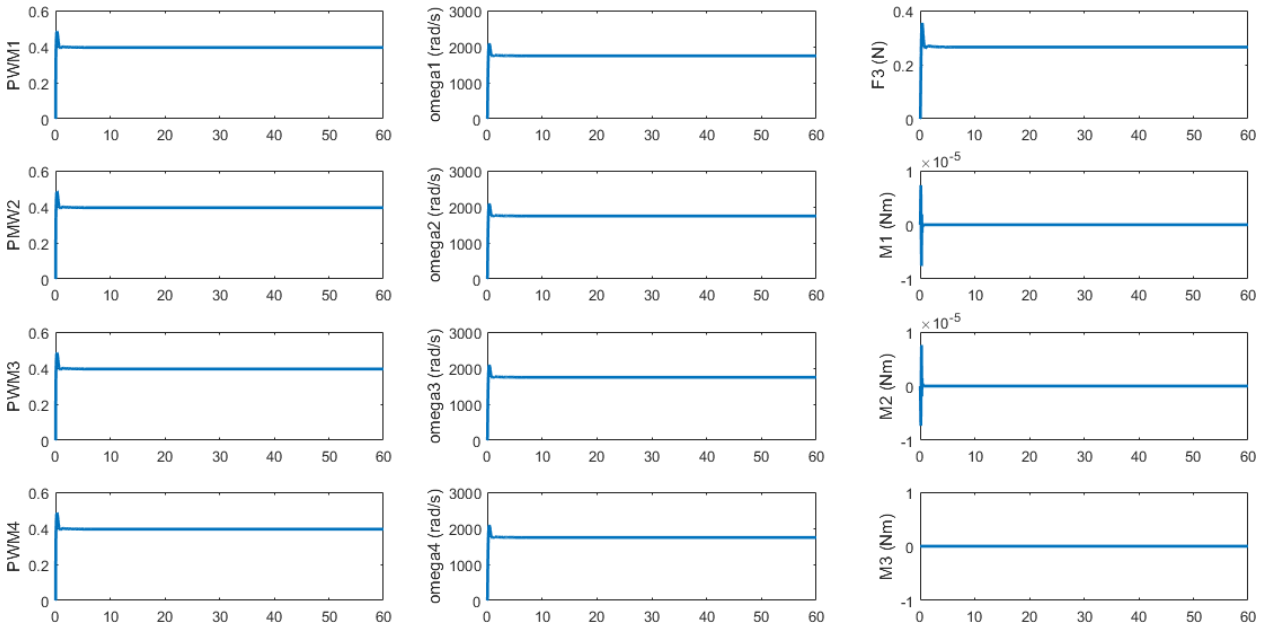
V07 (desplazamiento horizontal corto sentido negativo un eje):



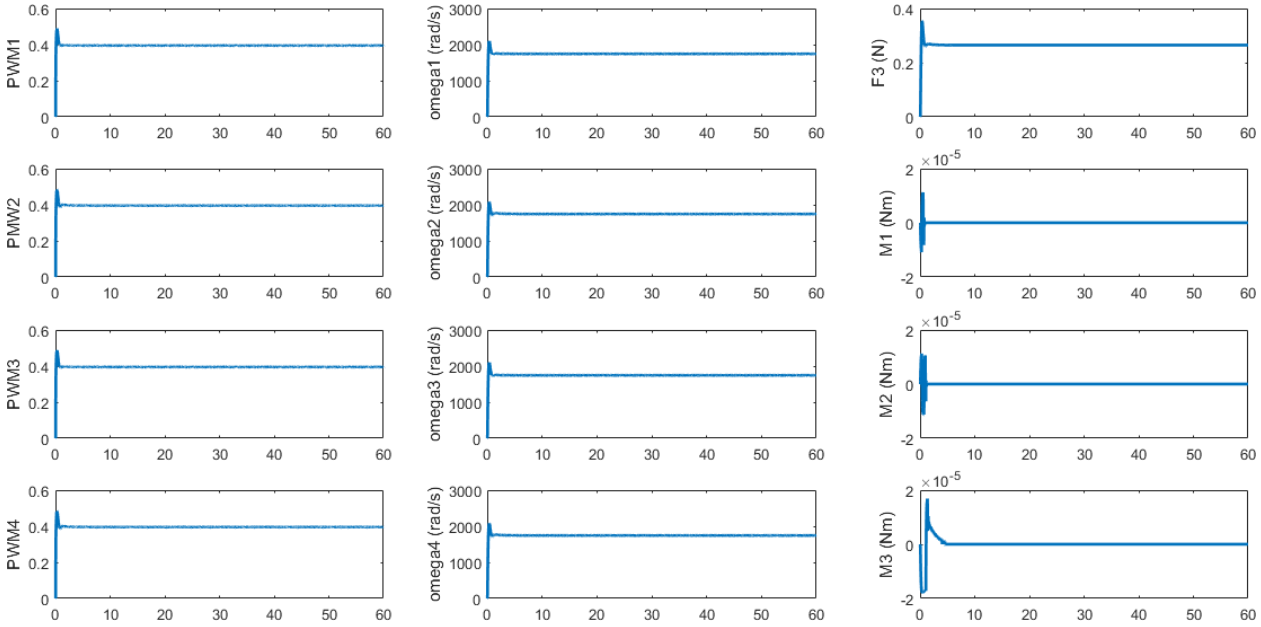
V08 (suelo inexistente, desplazamiento horizontal negativo dos ejes no simétrico):



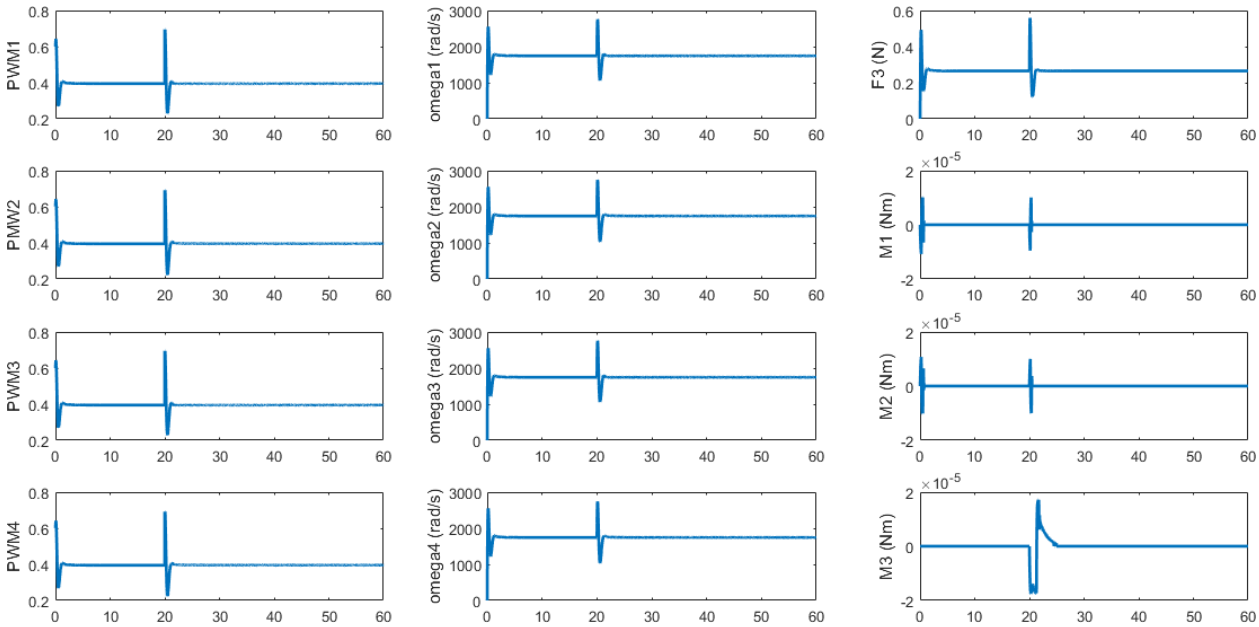
V09 (liberación en altura estabilizada con descenso en retroceso y estabilización):



V10 (liberación en altura no estabilizada, descenso en avance y rotación):



V11 (liberación en altura no estabilizada, aterrizaje, despegue en avance y guiñada):



ANEXO 5 Parámetros obtenidos por telemetría

Los datos de vuelo se obtienen por telemetría. Se ha configurado el cliente de la estación de tierra para que capture los ocho grupos de parámetros mostrados en la Tabla 28.

Tabla 28 : Grupos de parámetros obtenidos por telemetría

Grupo	Instrucciones captura parámetros	Ref.

REF	`LOG_GROUP_START(sys)`
	`LOG_ADD(LOG_INT8, canfly, &canFly)`
	`LOG_GROUP_STOP(sys)`
	
	`LOG_GROUP_START(ctrltarget)`
	`LOG_ADD(LOG_FLOAT,roll,&setpoint.attitude.roll)`
	`LOG_ADD(LOG_FLOAT, pitch, &setpoint.attitude.pitch)`
	`LOG_ADD(LOG_FLOAT, yaw, &setpoint.attitudeRate.yaw)`
	`LOG_GROUP_STOP(ctrltarget)`
		`system.c`
		
		`stabilizer.c`

SEN1	`LOG_GROUP_START(acc)`
	`LOG_ADD(LOG_FLOAT, x, &sensorData.acc.x)`
	`LOG_ADD(LOG_FLOAT, y, &sensorData.acc.y)`
	`LOG_ADD(LOG_FLOAT, z, &sensorData.acc.z)`
	`LOG_GROUP_STOP(acc)`
	
	`LOG_GROUP_START(baro)`
	`LOG_ADD(LOG_FLOAT, asl, &sensorData.baro.asl)`
	`LOG_ADD(LOG_FLOAT, temp, sensorData.baro.temperature)`
	`LOG_ADD(LOG_FLOAT, pressure, &sensorData.baro.pressure)`
	`LOG_GROUP_STOP(baro)`
		`stabilizer.c`
		
		`stabilizer.c`

SEN2	`LOG_GROUP_START(gyro)`
	`LOG_ADD(LOG_FLOAT, x, &sensorData.gyro.x)`
	`LOG_ADD(LOG_FLOAT, y, &sensorData.gyro.y)`
	`LOG_ADD(LOG_FLOAT, z, &sensorData.gyro.z)`
	`LOG_GROUP_STOP(gyro)`
	
	`LOG_GROUP_START(mag)`
	`LOG_ADD(LOG_FLOAT, x, &sensorData.mag.x)`
	`LOG_ADD(LOG_FLOAT, y, &sensorData.mag.y)`
	`LOG_ADD(LOG_FLOAT, z, &sensorData.mag.z)`
	`LOG_GROUP_STOP(mag)`
		`stabilizer.c`
		
		`stabilizer.c`

STA	`LOG_GROUP_START(stabilizer)`
	`LOG_ADD(LOG_FLOAT, roll, &state.attitude.roll)`
	`LOG_ADD(LOG_FLOAT, pitch, &state.attitude.pitch)`
	`LOG_ADD(LOG_FLOAT, yaw, &state.attitude.yaw)`
	`LOG_ADD(LOG_UINT16, thrust, &control.thrust)`
	`LOG_GROUP_STOP(stabilizer)`
		`stabilizer.c`

LOC1	`LOG_GROUP_START(controller)`
	`LOG_ADD(LOG_INT16, ctr_yaw, &control.yaw)`
	`LOG_GROUP_STOP(controller)`
	
	`LOG_GROUP_START(posEstimatorAlt)`
	`LOG_ADD(LOG_FLOAT, estimatedZ, &state.estimatedZ)`
	`LOG_ADD(LOG_FLOAT, velocityZ, &state.velocityZ)`
	`LOG_GROUP_STOP(posEstimatorAlt)`
	
	`LOG_GROUP_START(controller)`
	`LOG_ADD(LOG_FLOAT, actuatorThrust, &actuatorThrust)`
	`LOG_ADD(LOG_FLOAT, roll, &attitudeDesired.roll)`
	`LOG_ADD(LOG_FLOAT, pitch, &attitudeDesired.pitch)`
	`LOG_ADD(LOG_FLOAT, yaw, &attitudeDesired.yaw)`
	`LOG_GROUP_STOP(controller)`
		`stabilizer.c`
		
		`position_estimato`
		`r_altitude.c`
		
		`controller_pid.c`

LOC2	`LOG_GROUP_START(posCtlAlt)`	`position_controll`

	LOG_ADD(LOG_FLOAT, targetX, &this.pidX.setpoint) LOG_ADD(LOG_FLOAT, targetY, &this.pidY.setpoint) LOG_ADD(LOG_FLOAT, targetZ, &this.pidZ.setpoint) LOG_ADD(LOG_FLOAT, p, &this.pidZ.pid.outP) LOG_ADD(LOG_FLOAT, i, &this.pidZ.pid.outI) LOG_ADD(LOG_FLOAT, d, &this.pidZ.pid.outD) LOG_GROUP_STOP(posCtlAlt)	er_pid.c
MOT1	LOG_GROUP_START(motor) LOG_ADD(LOG_INT32, m4, &motorPower.m4) LOG_ADD(LOG_INT32, m1, &motorPower.m1) LOG_ADD(LOG_INT32, m2, &motorPower.m2) LOG_ADD(LOG_INT32, m3, &motorPower.m3) LOG_GROUP_STOP(motor)	power_distributio n_stock.c
MOT2	LOG_GROUP_START(pwm) LOG_ADD(LOG_UINT32, m1_pwm, &motor_ratios[0]) LOG_ADD(LOG_UINT32, m2_pwm, &motor_ratios[1]) LOG_ADD(LOG_UINT32, m3_pwm, &motor_ratios[2]) LOG_ADD(LOG_UINT32, m4_pwm, &motor_ratios[3]) LOG_GROUP_STOP(pwm)	motors.c

El cliente se configura directamente en su interface para que capture y registre los parámetros seleccionados, permitiéndonos fijar asimismo la frecuencia de muestreo de los mismos. Por otro lado, la configuración de los grupos de parámetros elegidos se guarda en archivos JSON para no tener que configurar el cliente cada vez que vamos a realizar un vuelo de prueba. Se muestra a continuación el archivo correspondiente al grupo REF que registra la bandera de paso de testeo inicial del vehículo y la señal de referencia de alabeo, cabeceo y guiñada. Los archivos correspondientes al resto de grupos son similares.

REF.json

```
{
  "logconfig": {
    "logblock": {
      "name": "REF",
      "period": 100,
      "variables": [
        {
          "name": "ctrltarget.pitch",
          "stored_as": "",
          "fetch_as": "float",
          "type": "TOC"
        },
        {
          "name": "ctrltarget.roll",
          "stored_as": "",
          "fetch_as": "float",
          "type": "TOC"
        },
        {
          "name": "ctrltarget.yaw",
          "stored_as": "",
          "fetch_as": "float",
          "type": "TOC"
        },
        {
          "name": "sys.canfly",
          "stored_as": "",
          "fetch_as": "int8_t",
          "type": "TOC"
        }
      ]
    }
  }
}
```

ANEXO 6 Archivos de tratamiento datos vuelo

Se reproducen a continuación todos los archivos que constituyen el paquete de tratamiento de datos de vuelo obtenidos por telemetría. El archivo principal (**`CrazyTelemetry_v0.m`**) contiene las instrucciones necesarias para su uso.

NOTA: los ocho archivos de importación que automatizan la conversión de los datos de vuelo en .csv a estructuras de datos los ha generado de forma automática MATLAB®, usándose con mínimos cambios relativos a la conversión de unidades, mientras que el archivo principal de tratamiento lo ha escrito en su totalidad el autor del presente trabajo.

Listado

```
CrazyTelemetry_v0.m
importfileLOC1.m
importfileLOC2.m
importfileMOT1.m
importfileMOT2.m
importfileREF.m
importfileSEN1.m
importfileSEN2.m
importfileSTA.m
```

`CrazyTelemetry_v0.m`

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                              CRAZYTELEMETRY V.0
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   Código para realizar de forma automatizada la captura y procesado de
%   los datos de telemetría generados durante los ensayos de vuelo
%   del Crazyflie 2.0
%   Alumno: Pablo Antón Jornet
%   Asignatura: TFM (Crazyflie 2.0) Curso 2016-17
%   Profesor: José Manuel Díaz Martínez
%   Máster Universitario en ingeniería de sistemas y de control
%   Universidad Nacional de Educación a Distancia
%   Programa: MATLAB 9.1.0.441655 (R2016b) Student Version
%   Llamadas a archivos externos: importfileLOC1.m, importfileLOC2.m
%   importfileMOT1.m, impotfileMOT2.m, importfileREF.m, importfileSEN1.m
%   importfileSEN2.m, importfileSTA.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   INSTRUCCIONES:
%   1) Situar todas las carpetas generadas por el cliente del Crazyflie 2.0
%   durante los vuelos de prueba (se genera una carpeta por vuelo)
%   en una carpeta del ordenador e introducir la ruta completa en la
%   variable ruta.
%   2) Ejecutar el script. Puede tardar en completar el proceso si hay
%   muchas carpetas para procesar
%   3) Se genenerará un archivo para cada ensayo en vuelo que tiene por
%   nombre la fecha y hora del ensayo (aaaammddThh-mm-ss.mat)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

clc; clear all; close all
```

```
%   Ruta carpeta conteniendo datos telemetría ** INTRODUCIR**

ruta = 'C:/Users/Pablo/Desktop/Master/CF2_flight_test/';

%   Cálculo automatizado rutas carpetas y archivos ensayos en vuelo
barra = '/';
directorio_actual = pwd;
cd (ruta)
contenido = dir; % Contenido carpeta base (contiene varios ensayos)
numero_carpetas = sum (~cellfun('isempty',{contenido.name}));
for i=1:numero_carpetas
    if ~(contenido(i).name(1) == '.' | contenido(i).name (1) == '..')
        if contenido(i).isdir == 1
        enrutado = strcat (ruta,contenido(i).name);
        cd (enrutado);
        cont_car = dir; % Contenido carpeta ensayo en vuelo (1 ensayo)
        numero_archivos = sum (~cellfun('isempty',{cont_car.name}));
        for j=1:numero_archivos
            if ~(cont_car(j).name(1) == '.' | cont_car(j).name(1) == '..')
                if strfind(cont_car(j).name,'STA')== 1
                    rutaSTA = strcat (enrutado,barra, cont_car(j).name);
                end
                if strfind(cont_car(j).name,'SEN2')== 1
                    rutaSEN2 = strcat (enrutado,barra,cont_car(j).name);
                end
                if strfind(cont_car(j).name,'SEN1')== 1
                    rutaSEN1 = strcat (enrutado,barra,cont_car(j).name);
                end
                if strfind(cont_car(j).name,'REF')== 1
                    rutaREF = strcat (enrutado,barra,cont_car(j).name);
                end
                if strfind(cont_car(j).name,'MOT2')== 1
                    rutaMOT2 = strcat (enrutado,barra,cont_car(j).name);
                end
                if strfind(cont_car(j).name,'MOT1')== 1
                    rutaMOT1 = strcat (enrutado,barra,cont_car(j).name);
                end
                if strfind(cont_car(j).name,'LOC2')== 1
                    rutaLOC2 = strcat (enrutado,barra,cont_car(j).name);
                end
                if strfind(cont_car(j).name,'LOC1')== 1
                    rutaLOC1 = strcat (enrutado,barra,cont_car(j).name);
                end
            end
        end
    %   Captura datos
    telemetria.REF = importfileREF(rutaREF);
    telemetria.STA = importfileSTA(rutaSTA);
    telemetria.SEN1 = importfileSEN1(rutaSEN1);
    telemetria.SEN2 = importfileSEN2(rutaSEN2);
    telemetria.LOC1 = importfileLOC1(rutaLOC1);
    telemetria.LOC2 = importfileLOC2(rutaLOC2);
    telemetria.MOT1 = importfileMOT1(rutaMOT1);
    telemetria.MOT2 = importfileMOT2(rutaMOT2);

    % Grabación datos y limpieza
    cd (directorio_actual)
    archivo = contenido(i).name;
```

```
    save(archivo,'telemetria');
    cd (ruta)
    clear telemetria
        end
    end
end
%   Vuelta a directorio base y limpieza
cd(directorio_actual)
clc, clear all
```

importfileLOC1.m

```
function telemetria = importfileLOC1(archivo)
delimiter = ',';
startRow = 2;
endRow = inf;
formatSpec = '%f%f%f%f%f%f%f%f%[^\n\r]';
fileID = fopen(archivo,'r');
dataArray = textscan(fileID, formatSpec, endRow(1)-startRow(1)+1,...
    'Delimiter', delimiter, 'HeaderLines', startRow(1)-1,...
    'ReturnOnError', false, 'EndOfLine', '\r\n');
for block=2:length(startRow)
    frewind(fileID);
    dataArrayBlock = textscan(fileID, formatSpec,...
        endRow(block)-startRow(block)+1, 'Delimiter', delimiter,...
        'HeaderLines', startRow(block)-1, 'ReturnOnError',...
        false, 'EndOfLine', '\r\n');
    for col=1:length(dataArray)
        dataArray{col} = [dataArray{col};dataArrayBlock{col}];
    end
end
fclose(fileID);
Timestamp = (dataArray{:, 1}./1000); % Conversion a segundos
telemetria.controller.actuatorThrust = timeseries(dataArray{:, 2},...
    Timestamp);
telemetria.controller.ctr_yaw = timeseries(dataArray{:, 3},Timestamp);
telemetria.controller.pitch = timeseries(dataArray{:, 4},Timestamp);
telemetria.controller.roll = timeseries(dataArray{:, 5},Timestamp);
telemetria.controller.yaw = timeseries(dataArray{:, 6},Timestamp);
telemetria.posEstimatorAlt.estimatedZ = timeseries(dataArray{:, 7},...
    Timestamp);
telemetria.posEstimatorAlt.velocityZ = timeseries(dataArray{:, 8},...
    Timestamp);
end
```

importfileLOC2.m

```
function telemetria = importfileLOC2(archivo)
delimiter = ',';
startRow = 2;
endRow = inf;
formatSpec = '%f%f%f%f%f%f%f%[^\n\r]';
fileID = fopen(archivo,'r');
dataArray = textscan(fileID, formatSpec, endRow(1)-startRow(1)+1,...
    'Delimiter', delimiter, 'HeaderLines', startRow(1)-1,...
    'ReturnOnError', false, 'EndOfLine', '\r\n');
```

```
for block=2:length(startRow)
    frewind(fileID);
    dataArrayBlock = textscan(fileID, formatSpec,...
        endRow(block)-startRow(block)+1, 'Delimiter', delimiter,...
        'HeaderLines', startRow(block)-1, 'ReturnOnError',...
        false, 'EndOfLine', '\r\n');
    for col=1:length(dataArray)
        dataArray{col} = [dataArray{col};dataArrayBlock{col}];
    end
end
fclose(fileID);
Timestamp = (dataArray{:, 1}./1000); % Conversion a segundos
telemetria.posCtlAlt.d = timeseries(dataArray{:, 2},Timestamp);
telemetria.posCtlAlt.i = timeseries(dataArray{:, 3},Timestamp);
telemetria.posCtlAlt.p = timeseries(dataArray{:, 4},Timestamp);
telemetria.posCtlAlt.targetX = timeseries(dataArray{:, 5},Timestamp);
telemetria.posCtlAlt.targetY = timeseries(dataArray{:, 6},Timestamp);
telemetria.posCtlAlt.targetZ = timeseries(dataArray{:, 7},Timestamp);
end
```

importfileMOT1.m

```
function telemetria = importfileMOT1(archivo)
delimiter = ',';
startRow = 2;
endRow = inf;
formatSpec = '%f%f%f%f%f%[^\n\r]';
fileID = fopen(archivo,'r');
dataArray = textscan(fileID, formatSpec, endRow(1)-startRow(1)+1,...
    'Delimiter', delimiter, 'HeaderLines', startRow(1)-1,...
    'ReturnOnError', false, 'EndOfLine', '\r\n');
for block=2:length(startRow)
    frewind(fileID);
    dataArrayBlock = textscan(fileID, formatSpec,...
        endRow(block)-startRow(block)+1, 'Delimiter', delimiter,...
        'HeaderLines', startRow(block)-1, 'ReturnOnError',...
        false, 'EndOfLine', '\r\n');
    for col=1:length(dataArray)
        dataArray{col} = [dataArray{col};dataArrayBlock{col}];
    end
end
fclose(fileID);
Timestamp = (dataArray{:, 1}./1000); % Conversion a segundos
telemetria.motor.m1 = timeseries(dataArray{:, 2},Timestamp);
telemetria.motor.m2 = timeseries(dataArray{:, 3},Timestamp);
telemetria.motor.m3 = timeseries(dataArray{:, 4},Timestamp);
telemetria.motor.m4 = timeseries(dataArray{:, 5},Timestamp);
end
```

importfileMOT2.m

```
function telemetria = importfileMOT2(archivo)
delimiter = ',';
startRow = 2;
endRow = inf;
formatSpec = '%f%f%f%f%f%[^\n\r]';
fileID = fopen(archivo,'r');
dataArray = textscan(fileID, formatSpec, endRow(1)-startRow(1)+1,...
```

```
    'Delimiter', delimiter, 'HeaderLines', startRow(1)-1,...
    'ReturnOnError', false, 'EndOfLine', '\r\n');
for block=2:length(startRow)
    frewind(fileID);
    dataArrayBlock = textscan(fileID, formatSpec,...
        endRow(block)-startRow(block)+1, 'Delimiter', delimiter,...
        'HeaderLines', startRow(block)-1, 'ReturnOnError',...
        false, 'EndOfLine', '\r\n');
    for col=1:length(dataArray)
        dataArray{col} = [dataArray{col};dataArrayBlock{col}];
    end
end
fclose(fileID);
Timestamp = (dataArray{:, 1}./1000);% Conversion a segundos
maxPWM = 65536; % Conversión a tanto por uno
telemetria.pwm.m1_pwm = timeseries(dataArray{:, 2},Timestamp);
telemetria.pwm.m2_pwm = timeseries(dataArray{:, 3},Timestamp);
telemetria.pwm.m3_pwm = timeseries(dataArray{:, 4},Timestamp);
telemetria.pwm.m4_pwm = timeseries(dataArray{:, 5},Timestamp);
telemetria.pwm.m1_pwm = (telemetria.pwm.m1_pwm./maxPWM);
telemetria.pwm.m2_pwm = (telemetria.pwm.m2_pwm./maxPWM);
telemetria.pwm.m3_pwm = (telemetria.pwm.m3_pwm./maxPWM);
telemetria.pwm.m4_pwm = (telemetria.pwm.m4_pwm./maxPWM);
end
```

importfileREF.m

```
function telemetria = importfileREF(archivo)
delimiter = ',';
startRow = 2;
endRow = inf;
formatSpec = '%f%f%f%f%f%[^\n\r]';
fileID = fopen(archivo,'r');
dataArray = textscan(fileID, formatSpec, endRow(1)-startRow(1)+1,...
    'Delimiter', delimiter, 'HeaderLines', startRow(1)-1,...
    'ReturnOnError', false, 'EndOfLine', '\r\n');
for block=2:length(startRow)
    frewind(fileID);
    dataArrayBlock = textscan(fileID, formatSpec,...
        endRow(block)-startRow(block)+1, 'Delimiter', delimiter,...
        'HeaderLines', startRow(block)-1, 'ReturnOnError',...
        false, 'EndOfLine', '\r\n');
    for col=1:length(dataArray)
        dataArray{col} = [dataArray{col};dataArrayBlock{col}];
    end
end
fclose(fileID);
Timestamp = (dataArray{:, 1}./1000);% Conversion a segundos
telemetria.ctrltarget.pitch = timeseries(dataArray{:, 2},Timestamp);
telemetria.ctrltarget.roll = timeseries(dataArray{:, 3},Timestamp);
telemetria.ctrltarget.yaw = timeseries(dataArray{:, 4},Timestamp);
telemetria.sys.canfly = timeseries(dataArray{:, 5},Timestamp);
end
```

importfileSEN1.m

```
function telemetria = importfileSEN1(archivo)
delimiter = ',';
```

```
startRow = 2;
endRow = inf;
formatSpec = '%f%f%f%f%f%f%f%[^\n\r]';
fileID = fopen(archivo,'r');
dataArray = textscan(fileID, formatSpec, endRow(1)-startRow(1)+1,...
    'Delimiter', delimiter, 'HeaderLines', startRow(1)-1,...
    'ReturnOnError', false, 'EndOfLine', '\r\n');
for block=2:length(startRow)
    frewind(fileID);
    dataArrayBlock = textscan(fileID, formatSpec,...
        endRow(block)-startRow(block)+1, 'Delimiter', delimiter,...
        'HeaderLines', startRow(block)-1, 'ReturnOnError',...
        false, 'EndOfLine', '\r\n');
    for col=1:length(dataArray)
        dataArray{col} = [dataArray{col};dataArrayBlock{col}];
    end
end
fclose(fileID);
Timestamp = (dataArray{:, 1}./1000);% Conversion a segundos
telemetria.acc.x = timeseries(dataArray{:, 2},Timestamp);
telemetria.acc.y = timeseries(dataArray{:, 3},Timestamp);
telemetria.acc.z = timeseries(dataArray{:, 4},Timestamp);
telemetria.baro.asl = timeseries(dataArray{:, 5},Timestamp);
telemetria.baro.pressure = timeseries(dataArray{:, 6},Timestamp);
telemetria.baro.temp = timeseries(dataArray{:, 7},Timestamp);
end
```

importfileSEN2.m

```
function telemetria = importfileSEN2(archivo)
delimiter = ',';
startRow = 2;
endRow = inf;
formatSpec = '%f%f%f%f%f%f%f%[^\n\r]';
fileID = fopen(archivo,'r');
dataArray = textscan(fileID, formatSpec, endRow(1)-startRow(1)+1,...
    'Delimiter', delimiter, 'HeaderLines', startRow(1)-1,...
    'ReturnOnError', false, 'EndOfLine', '\r\n');
for block=2:length(startRow)
    frewind(fileID);
    dataArrayBlock = textscan(fileID, formatSpec,...
        endRow(block)-startRow(block)+1, 'Delimiter', delimiter,...
        'HeaderLines', startRow(block)-1, 'ReturnOnError',...
        false, 'EndOfLine', '\r\n');
    for col=1:length(dataArray)
        dataArray{col} = [dataArray{col};dataArrayBlock{col}];
    end
end
fclose(fileID);
Timestamp = (dataArray{:, 1}./1000);% Conversion a segundos
telemetria.gyro.x = timeseries(dataArray{:, 2},Timestamp);
telemetria.gyro.y = timeseries(dataArray{:, 3},Timestamp);
telemetria.gyro.z = timeseries(dataArray{:, 4},Timestamp);
telemetria.mag.x = timeseries(dataArray{:, 5},Timestamp);
telemetria.mag.y = timeseries(dataArray{:, 6},Timestamp);
telemetria.mag.z = timeseries(dataArray{:, 7},Timestamp);
end
```

importfileSTA.m

```
function telemetria = importfileSTA(archivo)
delimiter = ',';
startRow = 2;
endRow = inf;
formatSpec = '%f%f%f%f%f%[^\n\r]';
fileID = fopen(archivo,'r');
dataArray = textscan(fileID, formatSpec, endRow(1)-startRow(1)+1,...
    'Delimiter', delimiter, 'HeaderLines', startRow(1)-1,...
    'ReturnOnError', false, 'EndOfLine', '\r\n');
for block=2:length(startRow)
    frewind(fileID);
    dataArrayBlock = textscan(fileID, formatSpec,...
        endRow(block)-startRow(block)+1, 'Delimiter', delimiter,...
        'HeaderLines', startRow(block)-1, 'ReturnOnError',...
        false, 'EndOfLine', '\r\n');
    for col=1:length(dataArray)
        dataArray{col} = [dataArray{col};dataArrayBlock{col}];
    end
end
fclose(fileID);
Timestamp = (dataArray{:, 1}./1000);% Conversion a segundos
telemetria.stabilizer.pitch = timeseries(dataArray{:, 2},Timestamp);
telemetria.stabilizer.roll = timeseries(dataArray{:, 3},Timestamp);
%telemetria.stabilizer.thrust = timeseries(dataArray{:, 4},Timestamp);
telemetria.stabilizer.thrust = timeseries(((dataArray{:,
4})./1e5),Timestamp);
telemetria.stabilizer.yaw = timeseries(dataArray{:, 5},Timestamp);
end
```