



Universidad  
Politécnica  
de Madrid

ETSI SISTEMAS  
INFORMÁTICOS

Trabajo fin de grado · Grado en ingeniería de computadores · Curso 2019 – 2020

## ***Desarrollo de un simulador de multicopteros para ensayo de controladores multivariable***

**Autor:** Pablo Daniel Gómez Elices

**Directores:** Marina Pérez Jiménez – Norberto Cañas de Paz

## Índice

Índice de figuras .....	1
Índice de tablas.....	3
1    Resumen .....	4
2 <i>Abstract</i> .....	5
3    Introducción .....	6
4    Procedimientos y recursos .....	9
5    Desarrollo del proyecto .....	10
5.1    Introducción .....	10
5.2    Especificación .....	11
5.3    Diseño .....	12
5.3.1    Descomposición funcional.....	12
5.3.2    HRT-HOOD .....	30
5.4    Implementación.....	37
5.4.1    Estructura de archivos .....	38
5.4.2    Útiles.....	39
5.4.3    Maniobras.....	42
5.4.4    Controlador de vuelo.....	44
5.4.5    Actuadores.....	52
5.4.6    Multicóptero.....	58
5.4.7    Sensores.....	66
5.4.8 <i>Logging</i> .....	70
5.4.9    Configuración.....	73
5.5    Pruebas.....	77
5.5.1    Parámetros de configuración comunes a todas las pruebas.....	77
5.5.2    Parámetros de tiempo real.....	78
5.5.3    Prueba 1: Caída libre .....	80
5.5.4    Prueba 2: Despegue.....	81
5.5.5    Prueba 3: <i>Roll</i> .....	82
5.5.6    Prueba 4: <i>Pitch</i> .....	83
5.5.7    Prueba 5: <i>Yaw rate</i> .....	84
5.5.8    Prueba 6: Dos referencias de <i>roll</i> .....	85
5.5.9    Prueba 7: <i>Roll</i> dinámico.....	86
6    Reflexión sobre los aspectos sociales, ambientales, éticos y profesionales .....	87
7    Líneas de mejora .....	88
8    Conclusiones.....	89

9	Anexos .....	90
9.1	<i>Tests</i> de planificabilidad de la prueba 3 .....	90
9.2	<i>Script</i> para ejecutar los tests de planificabilidad de la prueba 3.....	91
9.3	Cálculo del tensor de inercia .....	92
9.4	Código de las pruebas .....	94
9.4.1	Prueba 1.....	94
9.4.2	Prueba 2.....	95
9.4.3	Prueba 3.....	96
9.4.4	Prueba 4.....	97
9.4.5	Prueba 5.....	98
9.4.6	Prueba 6.....	99
9.4.7	Prueba 7.....	101
10	Bibliografía.....	102

## Índice de figuras

Figure 1 - Visualizador .....	6
Figure 2 – Configuración del cuadricóptero .....	7
Figure 3 - Estructura del proyecto a nivel de sistema .....	7
Figura 4 - Marcos de referencia.....	10
Figura 5 - Bloques funcionales.....	12
Figura 6 - Multicóptero - Bloque funcional .....	13
Figura 7 - Cuadricóptero - Vista aérea.....	15
Figura 8 - Marco de referencia local - Vista aérea.....	15
Figura 9 - Marco de referencia local - Vista lateral .....	15
Figura 10 - Brazos - Vista aérea .....	16
Figura 11 - Brazos - Vista lateral .....	16
Figura 12 - Fuerzas - Vista lateral .....	17
Figura 13 - Actuadores - Bloque funcional .....	18
Figura 14 - Sistema de primer orden .....	19
Figura 15 - Sistemas de primer orden .....	20
Figura 16 - Sistema de primer orden - Retenedor de orden cero .....	20
Figura 17 - Sensores - Bloque funcional .....	22
Figura 18 - Controlador de vuelo - Bloque funcional .....	22
Figura 19 - Controlador de vuelo - División en bloques funcionales más pequeños .....	23
Figura 20 - PID analógico en forma paralela .....	24
Figura 21 – Integrador analógico.....	24
Figura 22 - Integrador discretizado .....	25
Figura 23 - Derivador analógico.....	26
Figura 24 - Derivador discretizado .....	26
Figura 25 - Controlador PID digital en forma paralela.....	27
Figura 26 - Controlador PID digital en forma paralela mejorado .....	28
Figura 27 - Diagrama de componentes .....	30
Figura 28 - Componente Maneuvering .....	31
Figura 29 - Componente Flight_Controller.....	31
Figura 30 – Interfaces References_Reader, References_Writer y References_Operations.....	32
Figura 31 – Componente Actuators.....	32
Figura 32 – Interfaces ESCs_Reader, ESCs_Writer y ESCs_Operations .....	33
Figura 33 - Interfaces Motors_Reader y Motors_Operations .....	33
Figura 34 - Componente Multicopter .....	34
Figura 35 - Interfaces Motion_Reader y Motion_Operations .....	34
Figura 36 - Componente Sensors.....	35
Figura 37 – Interfaces Absolute_Orientation_Sensor_Reader y Absolute_Orientation_Sensor_Operations..	35
Figura 38 - Componente Logging.....	36
Figura 39 – Resultados de la prueba 1 .....	80
Figura 40 - Resultados de la prueba 2 .....	81
Figura 41 - Resultados de la prueba 3 .....	82
Figura 42 - Resultados de la prueba 4 .....	83
Figura 43 - Resultados de la prueba 5 .....	84
Figura 44 - Resultados de la prueba 6 .....	85
Figura 45 - Resultados de la prueba 7 .....	86

Figura 46 - Cálculo del tensor de inercia .....	92
---	----

## Índice de tablas

Tabla 1 - Notación SNAME.....	10
Tabla 2 - Vectores de movimiento .....	11
Tabla 3 - Bloques funcionales.....	12
Tabla 4 - Parámetros comunes a todas las pruebas.....	77
Tabla 5 - Asignación de prioridades .....	78
Tabla 6 - Uso de objetos protegidos por tareas .....	78
Tabla 7 - Prioridades de techo.....	79
Tabla 8 - Asignación de parámetros .....	90

## 1 Resumen

En este proyecto, desarrollamos el prototipo inicial de un simulador de multicopteros en tiempo real para ensayar controladores de vuelo multivariable. El prototipo inicial nos permite simular y controlar un cuadricóptero simple. El diseño se compone de dos partes principales; el simulador del cuadricóptero y el controlador de vuelo. El primero se basa en las ecuaciones de movimiento de un sólido-rígido en 6 grados de libertad en donde el diseño de las fuerzas y momento ha sido simplificado. El controlador de vuelo imita, de manera simplificada, el comportamiento que se puede encontrar en la mayoría de los controladores de vuelo comerciales en el ámbito no profesional. Se describen algunos resultados de simulación para mostrar el comportamiento del controlador implementado.

## 2 *Abstract*

In this project, we develop an initial prototype of a real-time multicopter simulator for testing multivariable flight controllers. This initial prototype allows us to simulate and control a simple quadcopter. The design is composed of two main parts; the quadcopter simulator and the flight controller. The first is based on the equations of motion for a 6 DoF rigid-body where the force and torque design have been simplified. The flight controller resembles, in a simplified way, the behavior that can be found in most commercial, hobby-oriented flight controllers. Some simulation results are described to show the behavior of the implemented controller.



### 3 Introducción

En este apartado vamos a detallar los objetivos del proyecto y su estructura básica. Los objetivos a conseguir en el proyecto son principalmente dos; construir un prototipo funcional de un simulador en tiempo real de multicopteros que permita ensayar controladores de vuelo y conseguir extraer de éste los componentes y principios del controlador de vuelo que se puedan aplicar a un *drone* real.

Al tratarse de un simulador en tiempo real, se ha usado el lenguaje de programación Ada para su implementación. Esto se debe a las facilidades que ofrece este lenguaje para programar en tiempo real. En particular, se ha usado la metodología *HRT-HOOD*, presentada en [1], para modelar el sistema. A partir de este modelado, la traducción a Ada es sistemática. En esta traducción se ha usado el Perfil de Ravenscar [2], un subconjunto de las funcionalidades de tiempo real de Ada para la implementación de sistemas de tiempo real críticos.

De manera complementaria al simulador, se ha desarrollado una herramienta en Python para la visualización de datos recogidos por el simulador. De ahora en adelante nos referiremos a ésta como el visualizador. A continuación, se puede ver el aspecto de éste.

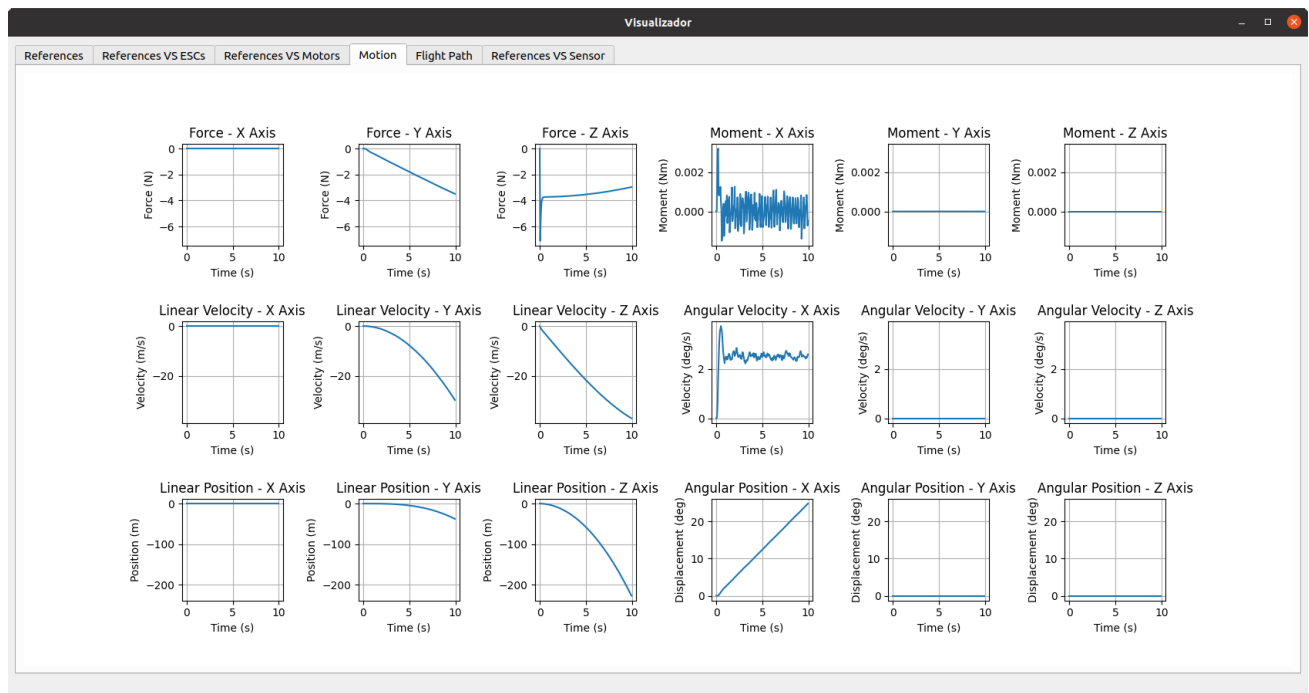


Figure 1 - Visualizador

En este prototipo inicial, sólo vamos a considerar un cuadricóptero con la configuración mostrada en la siguiente figura:

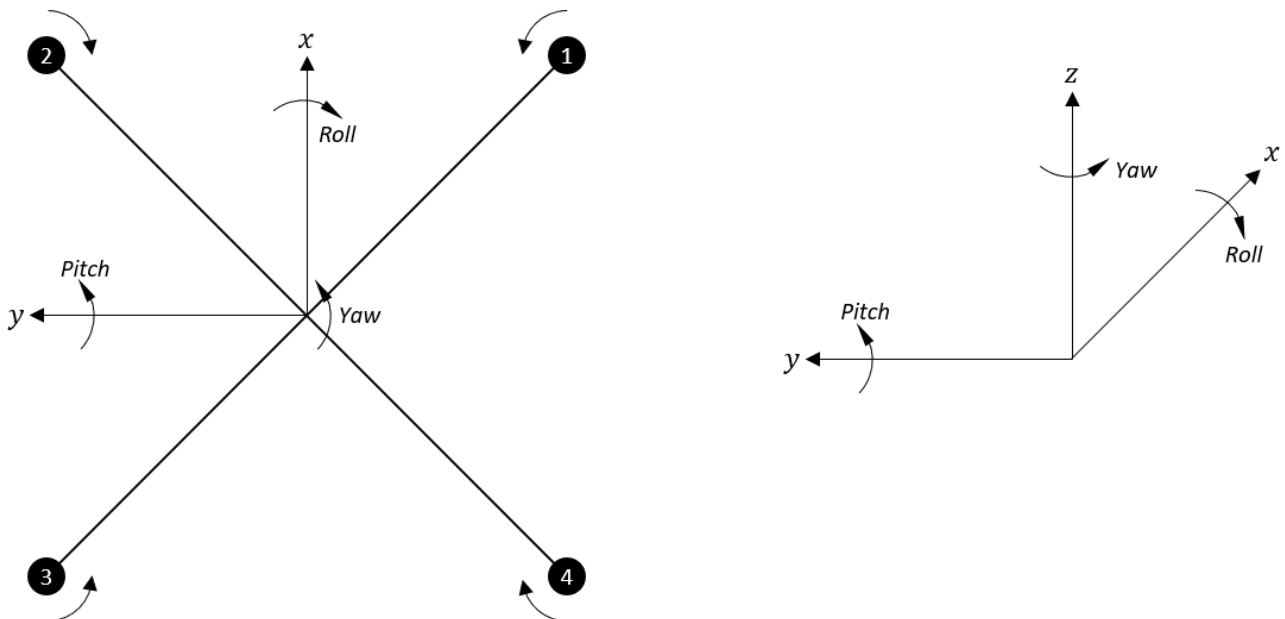


Figure 2 – Configuración del cuadricóptero

En la figura se detalla, a través de una vista aérea, la configuración de los brazos, la colocación de los ejes y la colocación de los motores con sus respectivos sentidos de giro. El sentido positivo del eje  $x$  define el frontal del *drone*. Una vez introducida esta configuración, podemos presentar la estructura del proyecto a nivel de sistema:

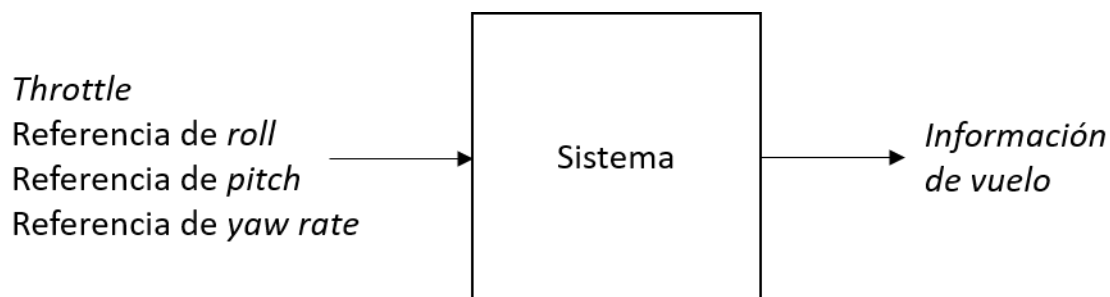


Figure 3 - Estructura del proyecto a nivel de sistema

Donde:

- **Throttle** es un valor en el rango  $[1000, 2000]$  que posteriormente se traducirá en la velocidad base de giro de los motores. Un valor de 1000 mantiene los motores parados, mientras que un valor de 2000 gira los motores a su máxima velocidad, todavía por determinar. Los valores de este rango no son arbitrarios. Históricamente los receptores por radio control que se usaban en configuraciones reales para controlar el *drone* enviaban al controlador de vuelo este valor codificado en una señal modulada por ancho de pulso, siendo su ancho pulso mínimo de  $1000 \mu s$  y su ancho de pulso máximo de  $2000 \mu s$ . En cuanto a la frecuencia de la señal, analizando un receptor real de este tipo, se ha obtenido una frecuencia aproximada de  $66.66 \text{ Hz}$ . Aunque en la actualidad muchos receptores no usen modulación por ancho de pulsos, se sigue usando como convenio este rango de valores.

- **Referencia de *roll***, o *roll* deseado ( $\phi_D$ ), es la rotación deseada en el eje  $x$ . Como veremos más adelante, y de manera general, asignar un valor positivo a  $\phi_D$  provocará que el *drone* se desplace hacia la derecha (sentido negativo del eje  $y$ ), mientras que un valor negativo hará que el *drone* se desplace hacia la izquierda (sentido positivo del eje  $y$ ).
- **Referencia de *pitch***, o *pitch* deseado ( $\theta_D$ ), es la rotación deseada en el eje  $y$ . Como veremos más adelante, y de manera general, asignar un valor positivo a  $\theta_D$  provocará que el *drone* se desplace hacia delante (sentido positivo del eje  $x$ ), mientras que un valor negativo hará que el *drone* se desplace hacia atrás (sentido negativo del eje  $x$ ).
- **Referencia de *yaw rate***, o *yaw rate* deseado ( $r_D$ ), es la velocidad de rotación deseada en el eje  $z$ . Como veremos más adelante, y de manera general, asignar un valor positivo a  $r_D$  provocará que el *drone* rote hacia la izquierda (sentido positivo de giro en el eje  $z$ ), mientras que un valor negativo hará que el *drone* rote hacia la derecha (sentido negativo de giro en el eje  $z$ ).
- **Información de vuelo** es toda aquella información relevante recogida durante el vuelo. Esta información se almacenará en un fichero con extensión *csv* (*comma-separated values*).

En el apartado 5 veremos en mayor detalle cómo usar estos valores de entrada para simular la respuesta del cuadricóptero y controlarlo. Antes de pasar a ver en detalle el desarrollo detallado, en el siguiente apartado veremos el *software* utilizado a lo largo del proyecto.

## 4 Procedimientos y recursos

En este apartado se enumera el *software* principal utilizado durante la realización del proyecto. Para la programación del simulador y del visualizador se han utilizado las siguientes herramientas:

- *GNAT Community 2020* para la programación del simulador en *Ada 2012*.
- *PyCharm Community 2020.2* para la programación del visualizador de datos en *Python 3*. También se han usado las siguientes librerías:
  - *pandas*.
  - *PyQt5*.
  - *Click*.
  - *Matplotlib*.
- *Git* y *GitLab* para control de versiones.
- *Ubuntu 20.04 LTS* como sistema operativo.
- *VMware Workstation 15 Player* para virtualizar *Ubuntu*.

Para la documentación se ha utilizado:

- *Microsoft Word* para la realización de este mismo documento.
- *Microsoft PowerPoint* para la creación de figuras.
- *Microsoft Visio* para la creación de diagramas UML.

De manera adicional, se ha usado *MATLAB R2020a* para el desarrollo de pequeños *scripts* y *Microsoft Excel* para la inspección de ficheros CSV.

## 5 Desarrollo del proyecto

### 5.1 Introducción

Empezamos introducción la notación y los convenios que se usarán a lo largo de todo este apartado. Al analizar el movimiento de móviles en 6 grados de libertad es conveniente definir dos marcos de referencia; el marco de referencia local y el marco de referencia global.

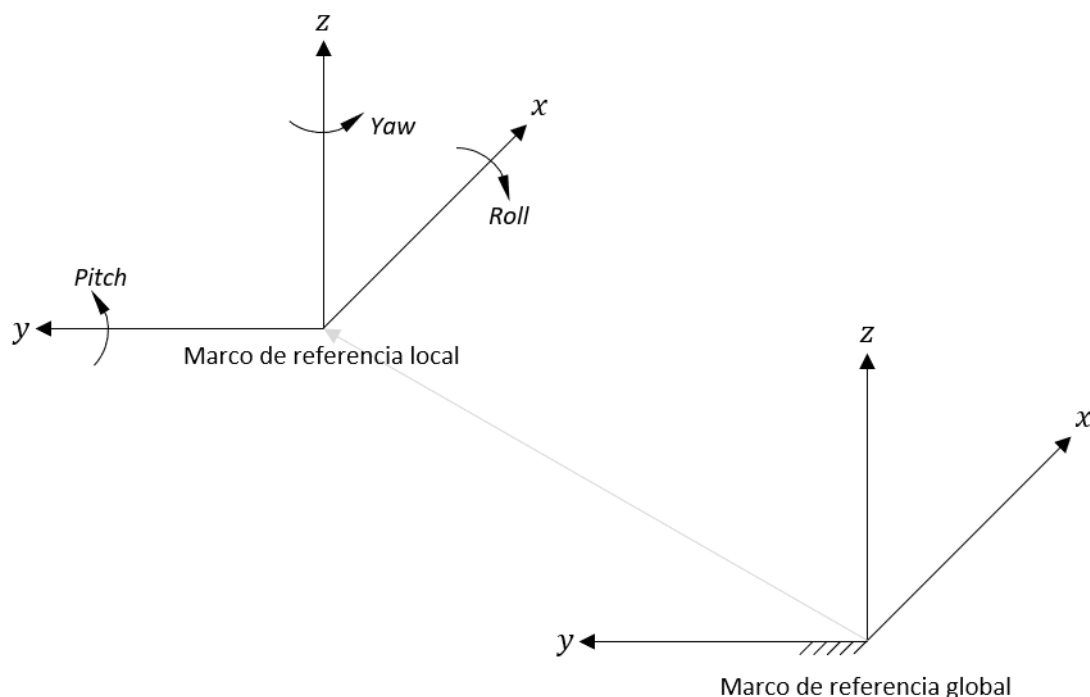


Figura 4 - Marcos de referencia

El marco de referencia local está fijado en el vehículo, generalmente en su centro de gravedad. Los movimientos de este marco de referencia local se describen en relación con el marco de referencia global, que permanece inmóvil en un punto fijo de la tierra. Para expresar las componentes de movimiento utilizaremos la notación SNAME.

Grados de libertad	Comentario	Fuerzas y momentos	Velocidades lineales y angulares	Posiciones y ángulos de Euler
1	Movimientos en la dirección $x$	$X$	$u$	$x$
2	Movimientos en la dirección $y$	$Y$	$v$	$y$
3	Movimientos en la dirección $z$	$Z$	$w$	$z$
4	Rotaciones alrededor del eje $x$	$K$	$p$	$\phi$ (Roll)
5	Rotaciones alrededor del eje $y$	$M$	$q$	$\theta$ (Pitch)
6	Rotaciones alrededor del eje $z$	$N$	$r$	$\psi$ (Yaw)

Tabla 1 - Notación SNAME

De esta manera, podemos describir el movimiento usando los siguientes vectores:

$\boldsymbol{\eta} = [\eta_1^T, \eta_2^T]^T$	$\boldsymbol{\eta}_1 = [x, y, z]^T$	$\boldsymbol{\eta}_2 = [\phi, \theta, \psi]^T$
$\boldsymbol{v} = [v_1^T, v_2^T]^T$	$\boldsymbol{v}_1 = [u, v, w]^T$	$\boldsymbol{v}_2 = [p, q, r]^T$
$\boldsymbol{\tau} = [\tau_1^T, \tau_2^T]^T$	$\boldsymbol{\tau}_1 = [X, Y, Z]^T$	$\boldsymbol{\tau}_2 = [K, M, N]^T$

Tabla 2 - Vectores de movimiento

Donde  $\boldsymbol{\eta}$  es el vector de posiciones y orientaciones en el marco de referencia global,  $\boldsymbol{v}$  es el vector de velocidades lineales y angulares en el marco de referencia local y  $\boldsymbol{\tau}$  es el vector de fuerzas y momentos en el marco de referencia local.

## 5.2 Especificación

Al tratarse el proyecto de un prototipo, vamos a definir únicamente, y de manera informal, los requisitos relevantes de cara al usuario final:

1. El simulador trabajará en tiempo real, es decir, para simular  $n$  segundos de vuelo, el tiempo de ejecución será de  $n + \epsilon$  segundos, dónde  $\epsilon$  es el tiempo empleado para la inicialización y terminación del simulador.
2. El simulador permitirá actualizar las referencias de vuelo en cualquier instante de la simulación.
3. El simulador escribirá, con un periodo configurable, los datos de vuelo en un fichero con extensión *csv* para su posterior visualización y análisis. Aunque algunas de estas cantidades no se han presentado todavía, se enumeran aquí por comodidad. Estos datos serán:
  - a. Sello de tiempo (*s*).
  - b. La referencia de *throttle* ( $\mu\text{s}$ ).
  - c. Las referencias de *roll* y *pitch* (*rad*).
  - d. La referencia de *yaw rate* (*rad/s*).
  - e. El ancho de pulso de los controladores electrónicos de velocidad (*ESCs*) ( $\mu\text{s}$ ).
  - f. Las velocidades angulares de los motores (*rad/s*).
  - g. Las fuerzas (*N*).
  - h. Los momentos (*N · m*).
  - i. Las velocidades lineales (*m/s*).
  - j. Las velocidades angulares (*rad/s*).
  - k. La posición (*m*).
  - l. La orientación (*rad*).
  - m. Las estimaciones del *roll* y el *pitch* (*rad*).
  - n. La estimación del *yaw rate* (*rad/s*).

### 5.3 Diseño

En este apartado vamos a describir cómo se ha obtenido el diagrama *HRT-HOOD* a partir de la descomposición funcional del proyecto.

#### 5.3.1 Descomposición funcional

Podemos descomponer el sistema en 4 bloques funcionales; *Controlador de vuelo*, *Actuadores*, *Multicóptero* y *Sensores*. La funcionalidad de la escritura de datos, por más intuitiva que la de los 4 bloques funcionales anteriores, se introducirá en el siguiente subapartado. A continuación, podemos ver la descripción de cada bloque y cómo interactúan entre ellos.

Bloque	Entradas	Salidas	Próposito
<b>Controlador de vuelo</b>	Referencias de vuelo y la estimación del <i>roll</i> , <i>pitch</i> y <i>yaw rate</i> .	Anchos de pulso de los ESCs.	Calcula los anchos de pulso de los ESCs a partir de la referencia de vuelo y de la estimación del <i>roll</i> , <i>pitch</i> y <i>yaw rate</i> .
<b>Actuadores</b>	Anchos de pulso de los ESCs.	Velocidades angulares de los motores.	Infiere las velocidades angulares de los motores a partir de los anchos de pulso de los ESCs.
<b>Multicóptero</b>	Velocidades angulares de los motores.	Velocidades angulares y orientación.	Calcula las fuerzas y momentos, velocidades lineales y angulares y la posición y orientación a partir de las velocidades angulares de los motores.
<b>Sensores</b>	Velocidades angulares y orientación.	La estimación del <i>roll</i> , <i>pitch</i> y <i>yaw rate</i> .	Infiere la estimación del <i>roll</i> , <i>pitch</i> y <i>yaw rate</i> a partir de las velocidades angulares y la orientación.

Tabla 3 - Bloques funcionales

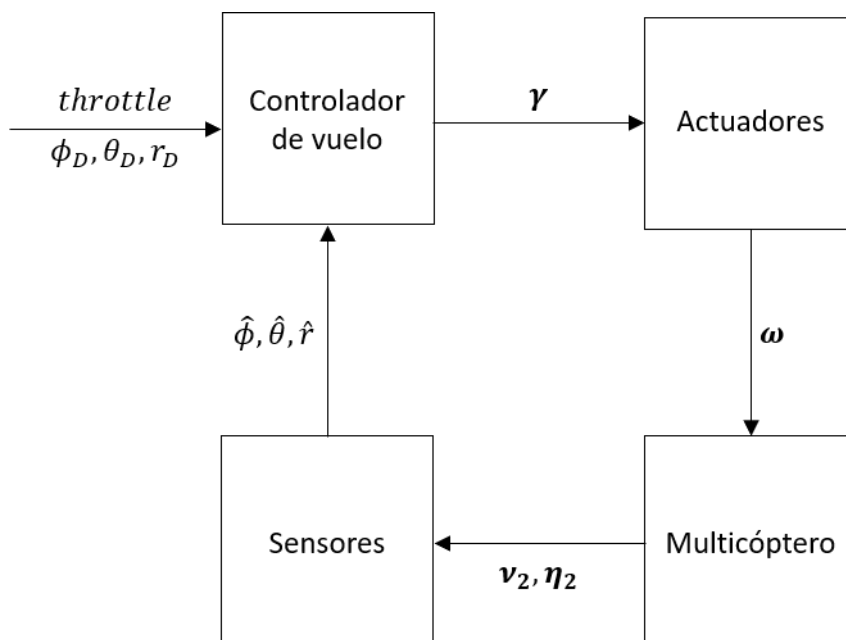


Figura 5 - Bloques funcionales

Antes de pasar a ver cada bloque funcional en detalle, vamos a brevemente introducir los parámetros y/o conceptos que nos faltan:

- La estimación del *roll*, *pitch* y *yaw rate*, respectivamente  $\hat{\phi}$ ,  $\hat{\theta}$  y  $\hat{r}$ , son los valores de *roll*, *pitch* y *yaw rate* que el sensor de orientación proporcionaría en una configuración real. Estas variables no deben ser confundidas con  $\phi$ ,  $\theta$  y  $r$ , que son valores teóricos calculados por el simulador.
- Los ESCs (controladores electrónicos de velocidad) son los dispositivos reales que controlan la velocidad de rotación de los motores. Son una combinación de *hardware* y *software* que ofrecen una interfaz de entrada basada en modulación por anchos de pulso y que se encargan de generar el tipo de señal adecuada para los motores. De ahora en adelante, supondremos que trabajan en el mismo rango que la referencia de *throttle* ( $[1000, 2000] \mu s$ ). De esta manera, un ancho de pulso de  $1000 \mu s$  mantendrá los motores parados, mientras que un valor de  $2000 \mu s$  hará que éstos giren a su máxima velocidad. Cada motor tiene su propio ESC asociado.  $\gamma = [\gamma_1, \gamma_2, \gamma_3, \gamma_4]$  es el vector de anchos de pulso.
- Las velocidades angulares de los motores ( $\omega = [\omega_1, \omega_2, \omega_3, \omega_4]$ ) se representan en *rad/s* y están en el rango  $[\omega_m, \omega_M]$  siendo  $\omega_m$  la velocidad angular mínima y  $\omega_M$  la máxima. El valor de estas constantes se detalla en el apartado de pruebas.

#### 5.3.1.1 Multicóptero

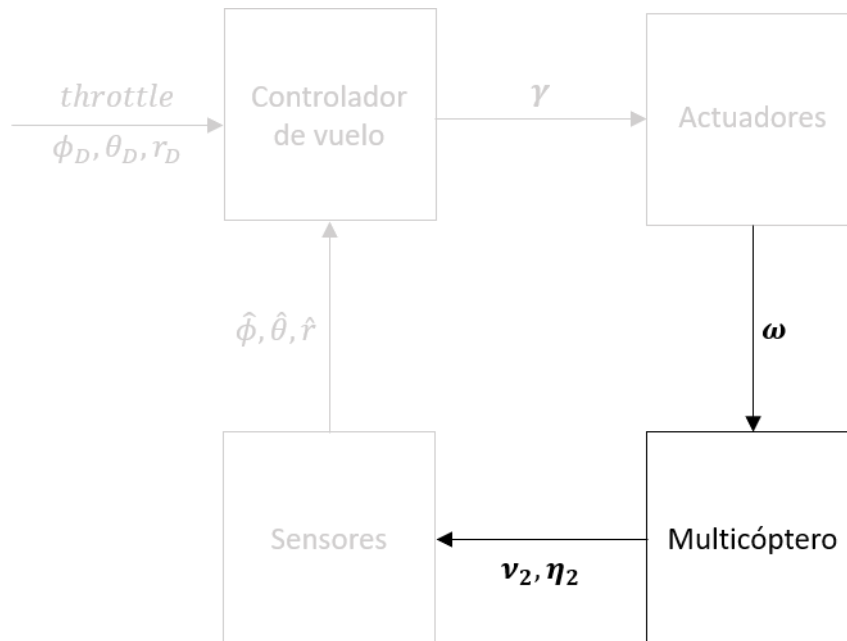


Figura 6 - Multicóptero - Bloque funcional

Las entradas son las velocidades angulares de los motores ( $\gamma$ ) y las salidas son los vectores  $\mathbf{v}_2$  y  $\boldsymbol{\eta}_2$ . Aunque las salidas sólo sean  $\mathbf{v}_2$  y  $\boldsymbol{\eta}_2$ , como veremos a continuación, es necesario calcular también  $\mathbf{v}_1$  y  $\boldsymbol{\eta}_1$ , siendo necesario  $\boldsymbol{\tau}$  para el cálculo de éstos. Suponiendo que el Multicóptero se comporta como un cuerpo rígido si ignoramos las deformaciones, podemos usar las ecuaciones de movimiento de un sólido-rígido con 6 grados de libertad. Estas ecuaciones se pueden encontrar en forma vectorial en [3]:

$$M_{RB}\dot{\mathbf{v}} + C_{RB}(\mathbf{v})\mathbf{v} = \boldsymbol{\tau}$$



Donde  $M_{RB}$  y  $C_{RB}(v)$  son:

$$M_{RB} = \begin{bmatrix} m & 0 & 0 & 0 & mz_G & -my_G \\ 0 & m & 0 & -mz_G & 0 & mx_G \\ 0 & 0 & m & my_G & -mx_G & 0 \\ 0 & -mz_G & my_G & I_x & -I_{xy} & -I_{xz} \\ mz_G & 0 & -mx_G & -I_{yx} & I_y & -I_{yz} \\ my_G & mx_G & 0 & -I_{zx} & -I_{zy} & I_z \end{bmatrix}$$

$$C_{RB}(v) = \begin{bmatrix} 0 & 0 & 0 & m(y_G q + z_G r) & -m(x_G q - w) & -m(x_G + v) \\ 0 & 0 & 0 & -m(y_G + w) & m(z_G r + x_G p) & -m(y_G r - u) \\ 0 & 0 & 0 & -m(z_G p - v) & -m(x_G q + u) & m(x_G p + y_G q) \\ -m(y_G q + z_G r) & m(y_G p + w) & m(z_G p - v) & 0 & -I_{yz} q - I_{xz} p + I_z r & I_{yz} r + I_{xy} p - I_y q \\ m(x_G q - w) & -m(z_G r + x_G p) & m(z_G q + u) & I_{yz} q + I_{xz} p - I_z r & 0 & -I_{xz} r - I_{xy} q + I_x p \\ m(x_G r + v) & m(y_G r - u) & -m(x_G p + y_G q) & -I_{yz} r - I_{xy} p + I_y q & I_{xz} r + I_{xy} q - I_z p & 0 \end{bmatrix}$$

Siendo  $m$  la masa del móvil,  $[x_G, y_G, z_G]$  el vector al centro de gravedad (desde el marco de referencia local) e  $I_{ij}$  la componente  $ij$  del tensor de inercia. Añadiendo la restricción de que el móvil sea simétrico ( $I_{xy} = I_{xz} = I_{yx} = I_{yz} = I_{zx} = I_{zy} = 0$ ) y que el origen del marco local sea el centro de gravedad ( $x_G = y_G = z_G = 0$ ), estas ecuaciones se pueden simplificar:

$$M_{RB} = \begin{bmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 \\ 0 & 0 & 0 & I_x & 0 & 0 \\ 0 & 0 & 0 & 0 & I_y & 0 \\ 0 & 0 & 0 & 0 & 0 & I_z \end{bmatrix}$$

$$C_{RB}(v) = \begin{bmatrix} 0 & 0 & 0 & 0 & mw & -mv \\ 0 & 0 & 0 & -mw & 0 & mu \\ 0 & 0 & 0 & mv & -mu & 0 \\ 0 & mw & -mv & 0 & I_z r & -I_y q \\ -mw & 0 & mu & -I_z r & 0 & I_x p \\ mv & -mu & 0 & I_y q & -I_z p & 0 \end{bmatrix}$$

También en [3] podemos encontrar las ecuaciones de la cinemática para un móvil con 6 grados de libertad:

$$\dot{\eta} = J(\eta)v$$

$$= \begin{bmatrix} J_1(\eta_2) & \mathbb{O}_{3 \times 3} \\ \mathbb{O}_{3 \times 3} & J_2(\eta_2) \end{bmatrix} v$$

Donde  $J_1(\eta_2)$  y  $J_2(\eta_2)$  son:

$$J_1(\eta_2) = \begin{bmatrix} c\psi c\theta & -s\psi c\phi + c\psi s\theta s\phi & s\psi s\phi + c\psi c\phi s\theta \\ s\psi c\theta & c\psi c\phi + s\psi s\theta s\phi & -c\psi s\phi + s\theta s\psi c\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix}$$

$$J_2(\eta_2) = \begin{bmatrix} 1 & s\phi t\theta & c\phi t\theta \\ 0 & c\phi & -s\phi \\ 0 & s\phi/c\theta & c\phi/c\theta \end{bmatrix}$$

Siendo  $s(\cdot) = \sin(\cdot)$ ,  $c(\cdot) = \cos(\cdot)$  y  $t(\cdot) = \tan(\cdot)$ . Cabe destacar que  $J_1(\eta_2)$  es una matriz de transformación del marco local al marco global que usaremos más adelante.

Observando estas ecuaciones se puede ver que a falta de fijar el valor de  $m$ ,  $I_x$ ,  $I_y$  y  $I_z$ , sólo falta por conocer el valor de  $\tau$ . Ignorando por ahora el efecto de la gravedad, el valor de  $\tau$  depende de la configuración geométrica del multicoptero y del comportamiento de los motores y de las hélices. A continuación, vamos a

derivar la expresión de  $\tau$  para la configuración ya presentada. Suponiendo el cuadricóptero de la siguiente figura, dónde se indican los sentidos de giro de cada motor,

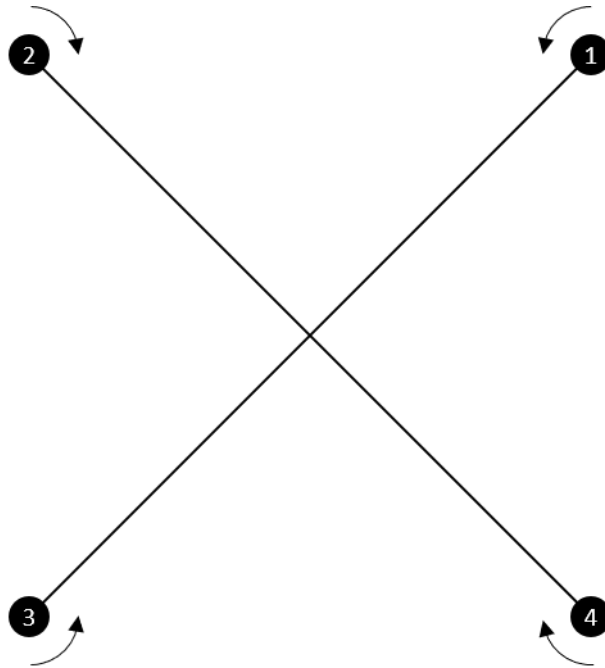


Figura 7 - Cuadricóptero - Vista aérea

el primer paso es colocar el marco de referencia local. El eje  $x$  define la parte frontal del cuadricóptero, mientras que en este caso el eje  $y$  apunta hacia el oeste del vehículo y el eje  $z$  apunta hacia arriba.

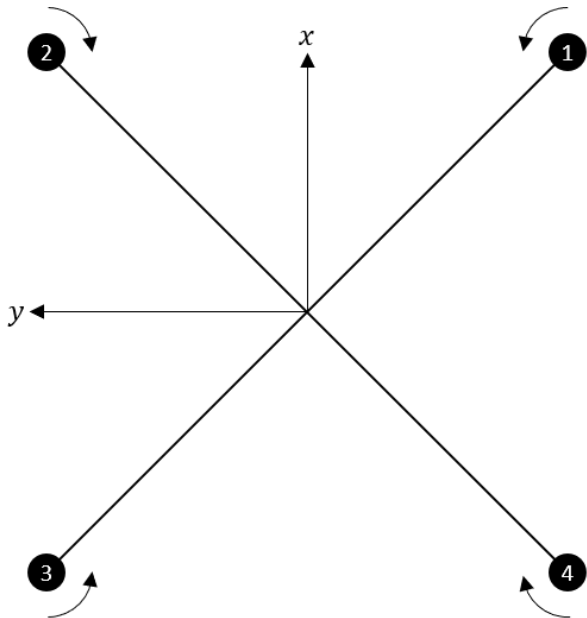


Figura 8 - Marco de referencia local - Vista aérea

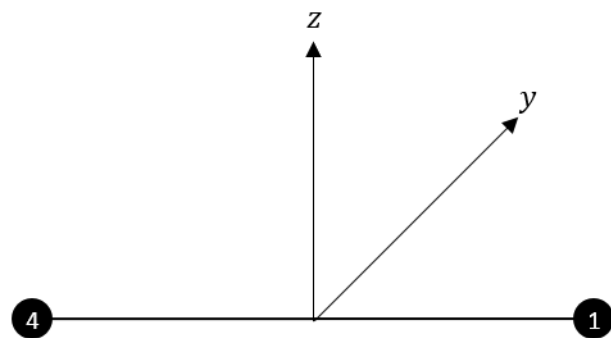


Figura 9 - Marco de referencia local - Vista lateral

El siguiente paso es calcular la distancia desde el centro de gravedad del vehículo hasta los motores. En este caso, fijando que los brazos miden  $l$ , y teniendo en cuenta que éstos están montados en cruz, podemos proyectar  $l$  a lo largo de los ejes  $x$  e  $y$ .

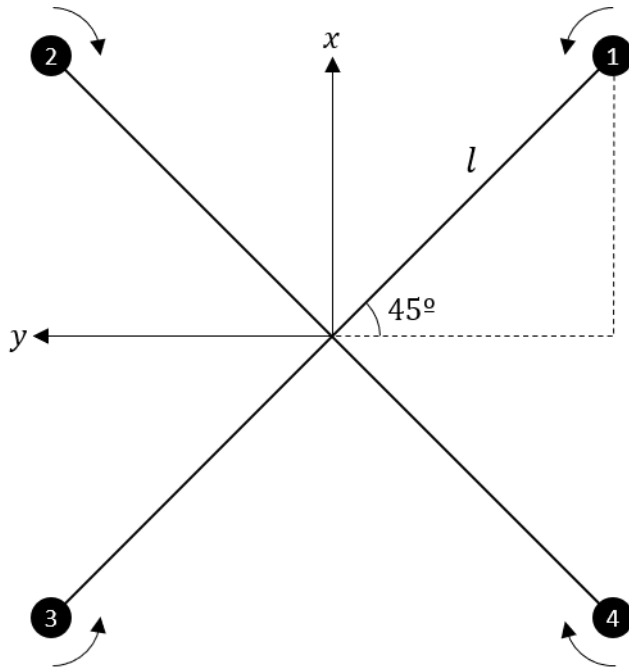


Figura 10 - Brazos - Vista aérea

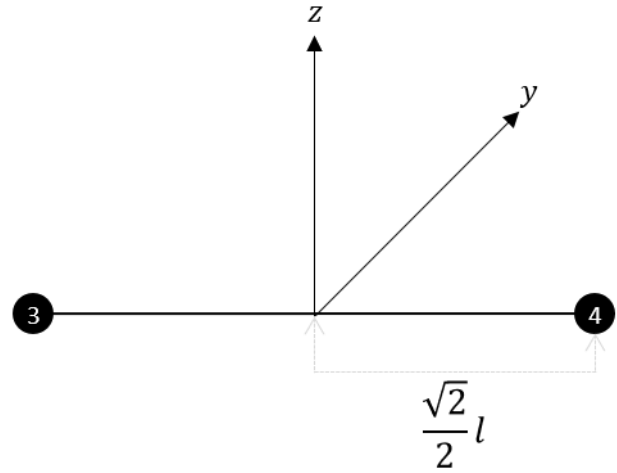


Figura 11 - Brazos - Vista lateral

Una vez fijada la configuración, analizamos el efecto producido sobre el vehículo al girar las hélices [4]. La velocidad angular de una hélice provoca una fuerza

$$f_{M_i} = b\omega_i^2$$

en la dirección positiva del eje  $z$  y un momento alrededor del eje  $z$

$$\tau_{M_i} = (-1)^i k\omega_i^2$$

Donde  $b$  y  $k$ , son dos parámetros, ambos mayores que 0, que dependen de la densidad del aire y del radio y forma de la hélice entre otros factores. Cabe destacar que la dirección del momento producido es la contraria a la dirección del giro de la hélice.

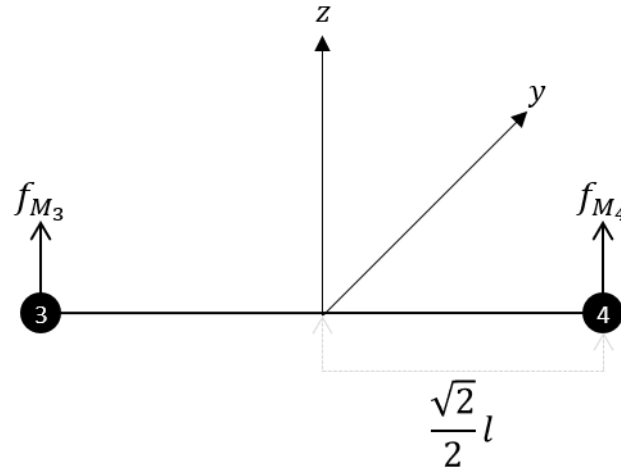


Figura 12 - Fuerzas - Vista lateral

Por tanto, sumando las contribuciones individuales de las hélices, tenemos que

$$Z = \sum_{i=1}^4 f_{M_i} = \sum_{i=1}^4 b\omega_i^2 = b \sum_{i=1}^4 \omega_i^2 = b(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2)$$

y

$$N = \sum_{i=1}^4 \tau_{M_i} = \sum_{i=1}^4 (-1)^i k\omega_i^2 = k \sum_{i=1}^4 (-1)^i \omega_i^2 = k(-\omega_1^2 + \omega_2^2 - \omega_3^2 + \omega_4^2)$$

Además, la fuerza ejercida por las hélices también contribuye a la rotación en los ejes  $x$  e  $y$ . En particular, las hélices 2 y 3 contribuyen a rotar positivamente alrededor del eje  $x$ , contribuyendo el resto de las hélices negativamente. En cuanto al eje  $y$ , las hélices 3 y 4 contribuyen a rotar positivamente alrededor de éste, contribuyendo el resto de las hélices negativamente:

$$K = \frac{\sqrt{2}}{2} lb(-\omega_1^2 + \omega_2^2 + \omega_3^2 - \omega_4^2)$$

$$M = \frac{\sqrt{2}}{2} lb(-\omega_1^2 - \omega_2^2 + \omega_3^2 + \omega_4^2)$$

No hay forma directa de aplicar fuerzas en los ejes  $x$  o  $y$ :

$$X = 0$$

$$Y = 0$$

Por tanto, ya tenemos la expresión inicial de  $\tau$ :

$$\tau = \begin{bmatrix} X \\ Y \\ Z \\ K \\ M \\ N \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ b & b & b & b \\ -\frac{\sqrt{2}}{2} lb & +\frac{\sqrt{2}}{2} lb & +\frac{\sqrt{2}}{2} lb & -\frac{\sqrt{2}}{2} lb \\ -\frac{\sqrt{2}}{2} lb & -\frac{\sqrt{2}}{2} lb & +\frac{\sqrt{2}}{2} lb & +\frac{\sqrt{2}}{2} lb \\ -k & +k & -k & +k \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix}$$

Para incorporar ahora la gravedad a  $\tau$  es suficiente con añadir la fuerza provocada por esta con respecto al marco de referencia local. Para ello basta con transformar el vector fuerza producido por la gravedad del marco global al marco local:

$$\tau = \begin{bmatrix} X \\ Y \\ Z \\ K \\ M \\ N \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ b & b & b & b \\ -\frac{\sqrt{2}}{2}lb & +\frac{\sqrt{2}}{2}lb & +\frac{\sqrt{2}}{2}lb & -\frac{\sqrt{2}}{2}lb \\ -\frac{\sqrt{2}}{2}lb & -\frac{\sqrt{2}}{2}lb & +\frac{\sqrt{2}}{2}lb & +\frac{\sqrt{2}}{2}lb \\ -k & +k & -k & +k \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} + \begin{bmatrix} J_1(\eta_2)^{-1} & \mathbb{O}_{3 \times 3} \\ \mathbb{O}_{3 \times 3} & \mathbb{O}_{3 \times 3} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} - mg \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Finalmente, utilizando el método de Euler [5] sobre las ecuaciones principales, conseguimos las expresiones discretas para poder implementar este bloque funcional:

$$\begin{aligned} \tau_{k+1} &= \dots \\ v_{k+1} &= v_k + h \left( M_{RB}^{-1} (\tau_k - C_{RBk}(v_k)v_k) \right) \\ \eta_{k+1} &= \eta_k + h(J(\eta_k)v_k) \end{aligned}$$

Siendo  $h$  el periodo con el que se actualicen estas expresiones.

#### 5.3.1.2 Actuadores

Las entradas son los anchos de pulso de los ESCs mientras que las salidas son las velocidades angulares de los motores.

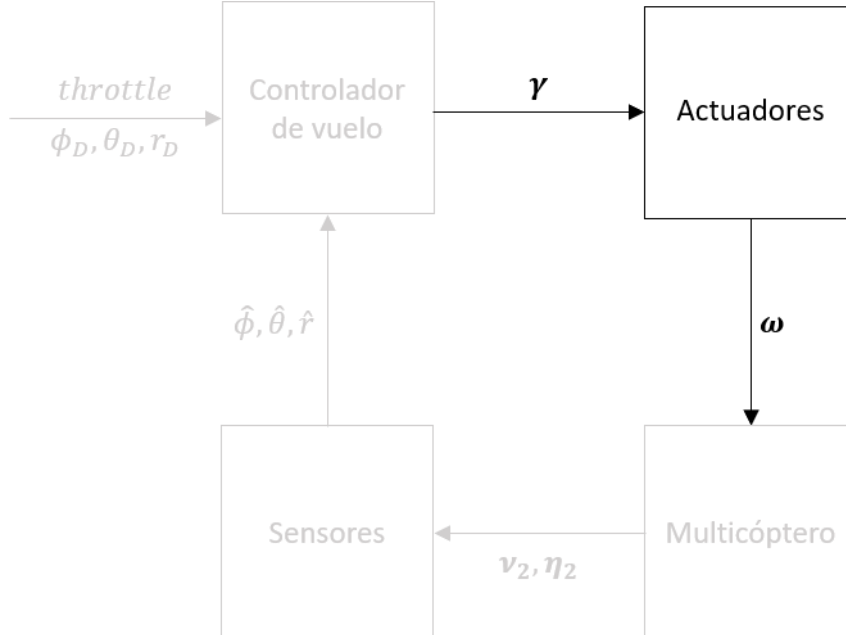


Figura 13 - Actuadores - Bloque funcional

Necesitamos establecer una relación entre los anchos de pulso de ESCs y las velocidades angulares de los motores:

$$\omega_i = f(\gamma_i)$$

Suponiendo que la velocidad angular crece linealmente variando el ancho de pulso, podemos aplicar la fórmula de la interpolación lineal [6]:

$$y = y_0 + (x - x_0) \frac{y_1 - y_0}{x_1 - x_0}$$

Podemos suponer que un ancho de pulso de  $1000 \mu s$  se convierte en una velocidad angular nula, es decir, mantiene los motores parados. Suponiendo ahora que un ancho de pulso de  $2000 \mu s$  se convierte en una velocidad angular máxima de  $\omega_M$ , tenemos que:

$$\omega_i = 0 + (\gamma_i - 1000) \frac{\omega_M - 0}{2000 - 1000} = (\gamma_i - 1000) \frac{\omega_M}{1000}$$

Cabe destacar que esta no es una aproximación demasiado realista pues se están obviando dos detalles:

- Estamos suponiendo que los ESCs puede hacer girar los motores a muy baja velocidad.
- Estamos suponiendo que la relación entre el ancho de pulso de los ESCs y la velocidad angular de los motores es lineal.

De todas formas, es suficiente para nuestros propósitos. La fórmula que hemos derivado para calcular (inferir) la velocidad angular podría implementarse directamente en *software* y que el cambio de velocidad fuese instantáneo. Para dar un enfoque un poco más realista, podemos considerar que la velocidad angular inferida se consigue alcanzar al cabo de un periodo de tiempo corto y no de forma inmediata. Este comportamiento se puede conseguir mediante un sistema de primer orden [7]. Podemos ver un sistema de primer orden en la siguiente figura.

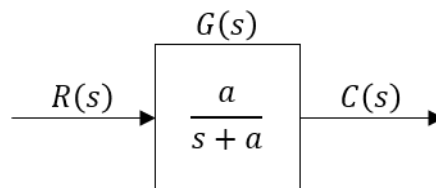


Figura 14 - Sistema de primer orden

En nuestro caso particular, la referencia sería la velocidad angular deseada y la salida sería la velocidad angular actual. La respuesta del sistema en el dominio de *Laplace* ante una entrada escalón es:

$$C(s) = R(s)G(s) = K \frac{1}{s} \cdot \frac{a}{s+a} = K \frac{a}{s(s+a)}$$

Aplicando la transformada inversa de Laplace, obtenemos la respuesta del sistema en el dominio del tiempo:

$$c(t) = K - K e^{-at} = K(1 - e^{-at})$$

En estos sistemas es usual definir el tiempo de estabilización (o *settling time*) como el tiempo que tarda la salida en alcanzar el 98% del valor de la referencia. Matemáticamente la expresión para este tiempo de estabilización se consigue despejando  $t$  de la expresión  $c(t) = 0.98K$ , obteniendo:

$$t_s = -\frac{\ln(0.02)}{a}$$

En la siguiente figura podemos encontrar las gráficas de  $c(t)$  para un escalón unitario con unos valores de  $t_s$  de 0.1 y de 0.250 segundos:

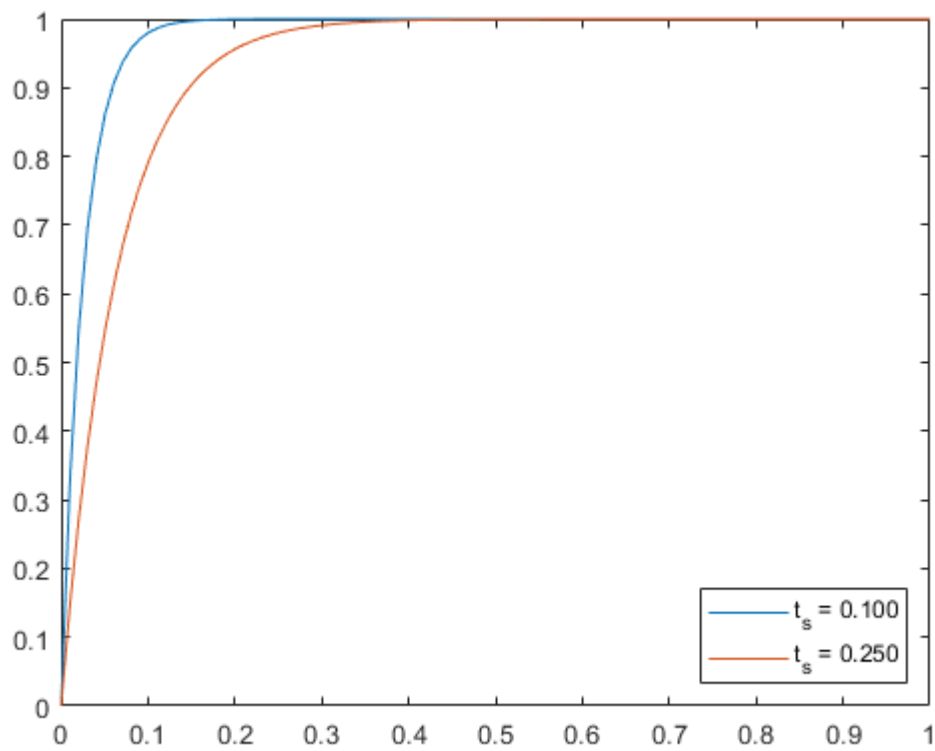


Figura 15 - Sistemas de primer orden

Para poder implementar en *software* el sistema de primer orden es necesario discretizarlo. Vamos a optar por el método del retenedor de orden cero [7].

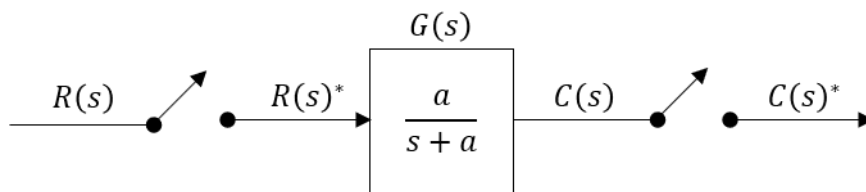


Figura 16 - Sistema de primer orden - Retenedor de orden cero

En [7] se puede encontrar el procedimiento para calcular  $G(z)$ :

$$\begin{aligned}
 G(z) &= \frac{z-1}{z} \mathbb{Z} \left\{ \frac{G(s)}{s} \right\} \\
 &= \frac{z-1}{z} \mathbb{Z} \left\{ \frac{\frac{a}{s+a}}{s} \right\} \\
 &= \frac{z-1}{z} \mathbb{Z} \left\{ \frac{a}{s(s+a)} \right\} \\
 &= \frac{z-1}{z} \cdot \frac{z(1-e^{-aT})}{(z-1)(z-e^{-aT})} \\
 &= \frac{1-e^{-aT}}{z-e^{-aT}} \\
 &= \frac{C(z)}{R(z)}
 \end{aligned}$$

Teniendo en cuenta ahora que  $G(z) = C(z)/R(z)$  podemos despejar  $C(z)$ :

$$\begin{aligned}
 \frac{C(z)}{R(z)} &= \frac{1-e^{-aT}}{z-e^{-aT}} \\
 \frac{C(z)}{R(z)} &= \frac{z^{-1}(1-e^{-aT})}{1-z^{-1}e^{-aT}} \\
 C(z)(1-z^{-1}e^{-aT}) &= R(z)(z^{-1}(1-e^{-aT})) \\
 C(z) - C(z)z^{-1}e^{-aT} &= R(z)z^{-1}(1-e^{-aT}) \\
 C(z) - e^{-aT}C(z)z^{-1} &= (1-e^{-aT})R(z)z^{-1} \\
 C(z) &= e^{-aT}C(z)z^{-1} + (1-e^{-aT})R(z)z^{-1}
 \end{aligned}$$

Y convirtiendo a ecuación en diferencias finalmente llegamos a la expresión que podemos implementar:

$$C_k = e^{-aT}C_{k-1} + (1-e^{-aT})R_{k-1}$$



### 5.3.1.3 Sensores

Las entradas son los vectores  $\mathbf{v}_2$  y  $\boldsymbol{\eta}_2$  mientras que las salidas son  $\hat{\phi}$ ,  $\hat{\theta}$  y  $\hat{r}$ . En el prototipo inicial, estas salidas son simplemente los valores muestreados de  $\phi$ ,  $\theta$  y  $r$  a una frecuencia más baja que la de la actualización de estos mismos valores en el bloque *Multicóptero*.

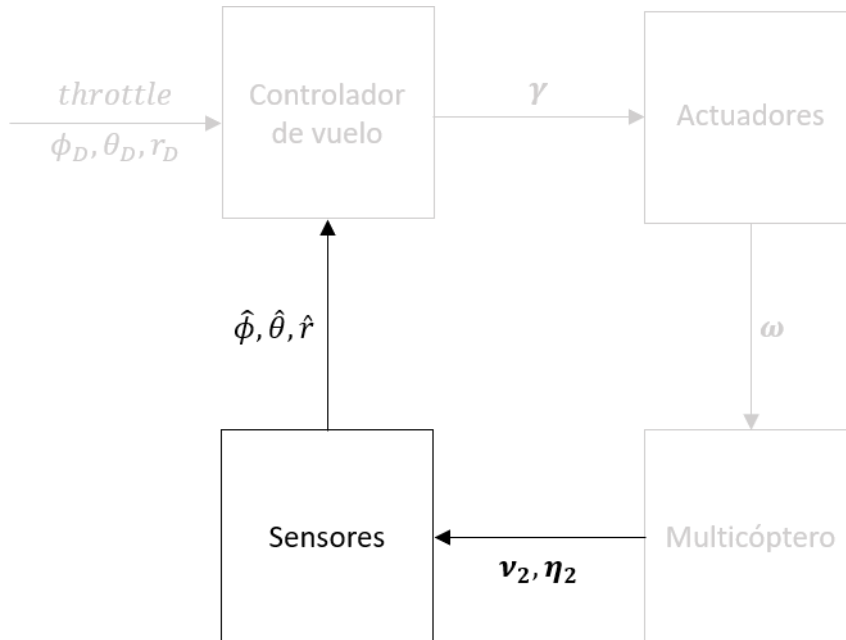


Figura 17 - Sensores - Bloque funcional

### 5.3.1.4 Controlador de vuelo

El controlador de vuelo se encarga de generar la señal adecuada para los actuadores.

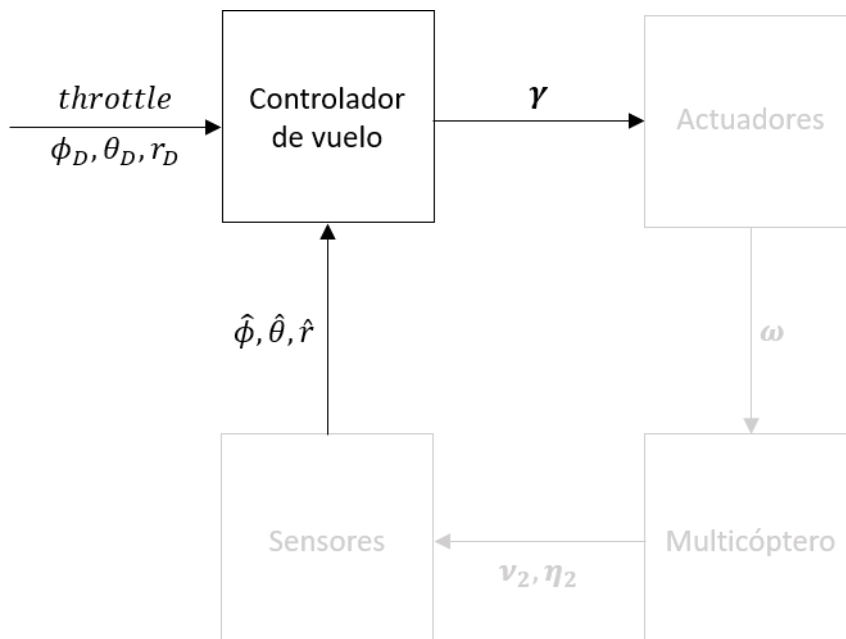


Figura 18 - Controlador de vuelo - Bloque funcional

Las entradas del controlador de vuelo son, por una parte, la referencia de posición angular en el eje  $x$  ( $\phi_D$ ), la referencia de posición angular en el eje  $y$  ( $\theta_D$ ) y la referencia de velocidad angular en el eje  $z$  ( $r_D$ ). Por otra parte, tenemos la estimación de posición angular en el eje  $x$  ( $\hat{\phi}$ ), la estimación de posición angular en el eje  $y$  ( $\hat{\theta}$ ) y la estimación de velocidad angular en el eje  $z$  ( $\hat{r}$ ).

La salida  $\gamma$  son los anchos de pulso para los ESCs. Este bloque funcional se puede dividir a su vez en otros bloques funcionales más pequeños:

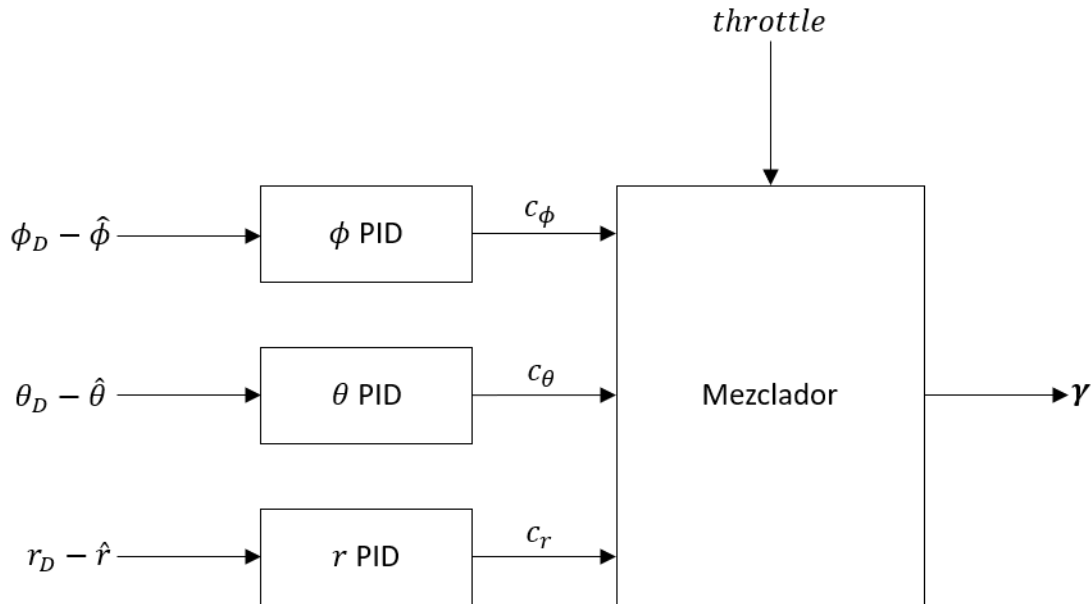


Figura 19 - Controlador de vuelo - División en bloques funcionales más pequeños

#### 5.3.1.4.1 Controladores PID

Como estrategia de control se opta por un planteamiento descentralizado, al ser diagonal la matriz de funciones de transferencia. Por ello, para obtener los anchos de pulso, se proponen 3 controladores *PID* que alimentamos con el error en el *roll*, en el *pitch* y en el *yaw rate*. Las salidas de estos controladores se mezclarán, junto con la referencia de *throttle*, en el mezclador. Éste también se encargará de recortar las señales mezcladas para que coincidan con el rango de los ESCs.

Para la implementación de los controladores PID usaremos una versión digital de un controlador PID analógico en forma paralela, que posteriormente discretizaremos. La ecuación de salida para este tipo de controladores es:

$$c(t) = K_P e(t) + K_I \int_0^t e(\tau) d\tau + K_D \frac{d}{dt} e(t)$$

En el dominio de Laplace, la función de transferencia es:

$$G(s) = K_P + K_I \frac{1}{s} + K_D s$$

Se puede ver el diagrama de bloques en la siguiente figura:

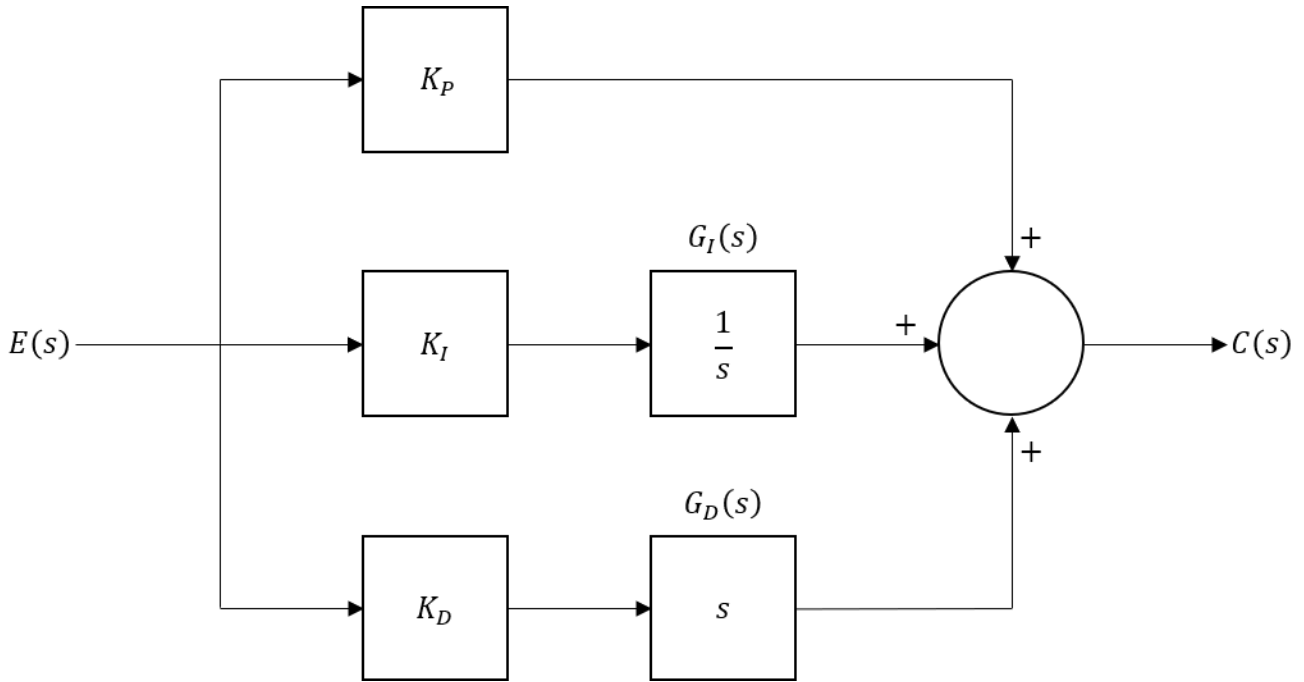


Figura 20 - PID analógico en forma paralela

Para discretizar el controlador vamos a discretizar de manera individual el bloque integrador y el bloque derivador. Vamos a empezar por el integrador.

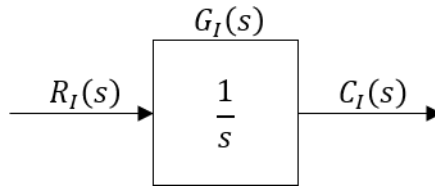


Figura 21 – Integrador analógico

Aplicando la transformada inversa de Laplace al bloque del integrador, obtenemos:

$$\begin{aligned}\mathcal{L}^{-1}\{C_I(s)\} &= \mathcal{L}^{-1}\{R_I(s)G_I(s)\} \\ &= \mathcal{L}^{-1}\left\{R_I(s)\frac{1}{s}\right\} \\ &= \int_{0^-}^t r_I(\tau) d\tau \\ &= c_I(t)\end{aligned}$$

Haciendo el cambio  $t = KT$ , tenemos que:

$$\begin{aligned}c_I(KT) &= \int_{0^-}^{KT} r_I(t) dt \\ &= \int_{0^-}^{KT-T} r_I(t) dt + \int_{KT-T}^{KT} r_I(t) dt \\ &= c_I(KT - T) + \int_{KT-T}^{KT} r_I(t) dt\end{aligned}$$

Cabe destacar que hasta este momento no hemos discretizado nada pues todavía no hemos impuesto que  $K$  sea un entero. Imponiendo ahora esa condición, podemos aproximar la integral usando la regla trapezoidal [8]:

$$\int_a^b f(x) dx \approx (b-a) \frac{f(b) + f(a)}{2}$$

Por tanto, obtenemos la siguiente ecuación discreta:

$$\begin{aligned} c_I(KT) &\approx c_I(KT-T) + (KT - (KT-T)) \cdot \frac{r_I(KT) + r_I(KT-T)}{2} \\ &= c_I(KT-T) + T \frac{r_I(KT) + r_I(KT-T)}{2} \\ &= c_I((K-1)T) + T \cdot \frac{r_I(KT) + r_I((K-1)T)}{2} \end{aligned}$$

Calculando la transformada Z, obtenemos:

$$\begin{aligned} C_I(z) &\approx C_I(z)z^{-1} + \frac{T}{2}(R_I(z) + R_I(z)z^{-1}) \\ &\approx C_I(z)z^{-1} + \frac{T}{2}R_I(z)(1+z^{-1}) \end{aligned}$$

Y finalmente, resolviendo para  $C_I(z)/R_I(z) = G_I(z)$ , tenemos que:

$$\begin{aligned} C_I(z) - C_I(z)z^{-1} &\approx \frac{T}{2}R_I(z)(1+z^{-1}) \\ C_I(z)(1-z^{-1}) &\approx \frac{T}{2}R_I(z)(1+z^{-1}) \\ \frac{C_I(z)}{R_I(z)} &\approx \frac{T}{2} \frac{1+z^{-1}}{1-z^{-1}} \\ G_I(z) &\approx \frac{T}{2} \frac{z+1}{z-1} \end{aligned}$$

Se puede observar que se llega a la misma expresión aplicando directamente la transformada bilineal:

$$G_I(z) \approx G_I(s) \Big|_{s=\frac{2}{T} \frac{z-1}{z+1}} = \frac{1}{\frac{2}{T} \frac{z-1}{z+1}} = \frac{T}{2} \frac{z+1}{z-1}$$

Ya tenemos el integrador discretizado:

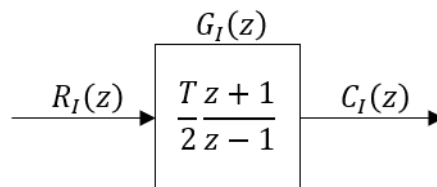


Figura 22 - Integrador discretizado

Para discretizar el derivador, usamos un enfoque parecido al anterior.

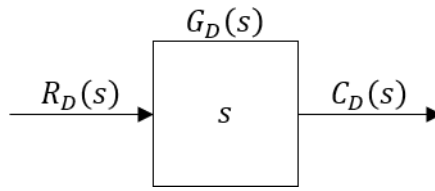


Figura 23 - Derivador analógico

Aplicando la transformada inversa de Laplace al bloque del derivador, obtenemos:

$$\begin{aligned}\mathcal{L}^{-1}\{C_D(s)\} &= \mathcal{L}^{-1}\{R_D(s)G_D(s)\} \\ &= \mathcal{L}^{-1}\{R_D(s)s\} \\ &= \frac{d}{dt}r_D(t) \\ &= c_D(t)\end{aligned}$$

Haciendo el cambio  $t = KT$ , tenemos que:

$$c_D(KT) = \frac{d}{dt}r_D(KT)$$

Si utilizamos la aproximación discreta de la derivada, llegamos a la siguiente ecuación discreta:

$$\begin{aligned}c_D(KT) &\approx \frac{r_D(KT) - r_D(KT - T)}{T} \\ &\approx \frac{r_D(KT) - r_D((K - 1)T)}{T}\end{aligned}$$

Calculando ahora la transformada Z, obtenemos:

$$\begin{aligned}C_D(z) &\approx \frac{R_D(z) - R_D(z)z^{-1}}{T} \\ &\approx \frac{R_D(z)(1 - z^{-1})}{T}\end{aligned}$$

Y finalmente resolviendo para  $C_D(z)/R_D(z) = G_D(z)$ , llegamos a:

$$\begin{aligned}\frac{C_D(z)}{R_D(z)} &\approx \frac{1 - z^{-1}}{T} \\ G_D(z) &\approx \frac{z - 1}{zT}\end{aligned}$$

Ya tenemos el derivador discretizado:

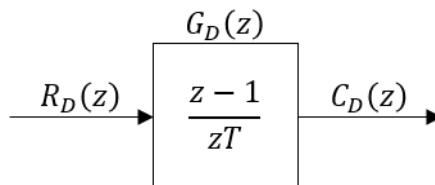


Figura 24 - Derivador discretizado

Por tanto, la planta en el dominio Z del controlador es:

$$G(z) = K_P + K_I \frac{Tz + 1}{2z - 1} + K_D \frac{z - 1}{zT}$$

A continuación, se puede ver el diagrama de bloques del PID digital:

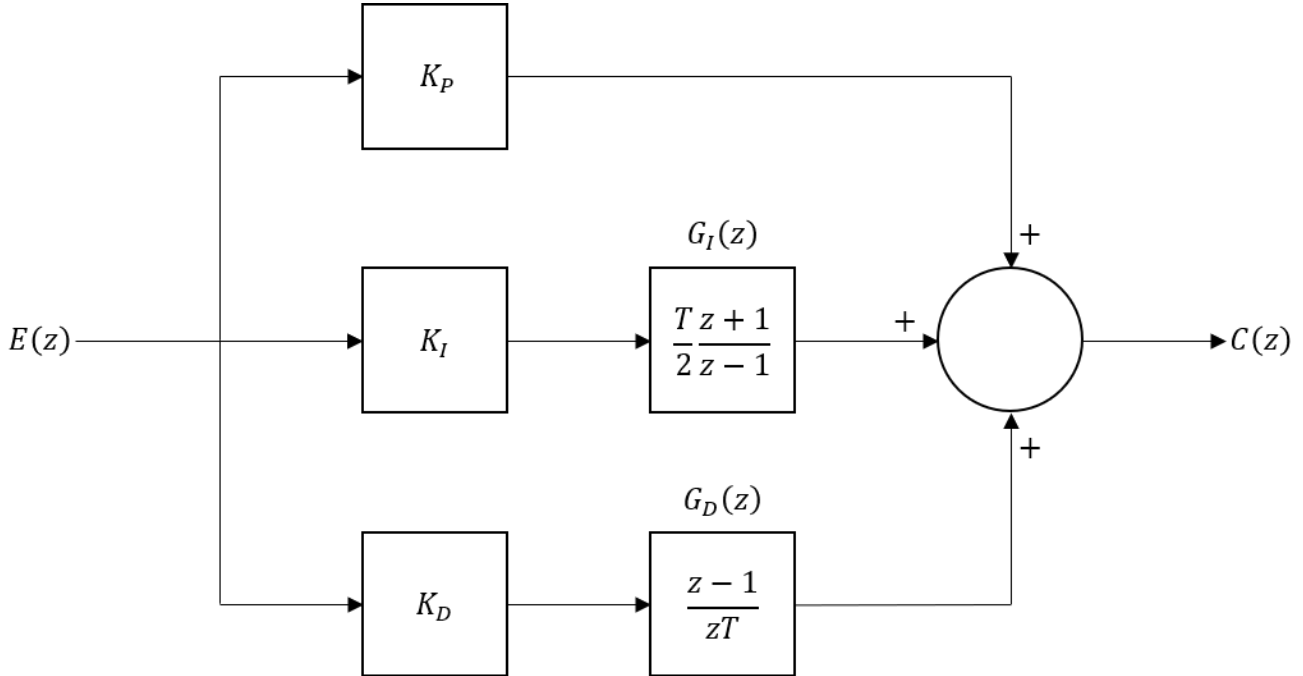


Figura 25 - Controlador PID digital en forma paralela

La obtención de este diagrama en el plano Z se muestra por completar la información, ya que de cara a la implementación sería suficiente con obtener las expresiones discretizadas de cada rama, obtenidas a partir del diagrama en el plano de Laplace, sin tener que usar la transformada Z. A continuación, mostramos cómo obtener las expresiones para la implementación a partir de este último diagrama. La salida del controlador es:

$$\begin{aligned} C(z) &= E(z)K_P + E(z)K_I \frac{Tz + 1}{2z - 1} + E(z)K_D \frac{z - 1}{zT} \\ &= C_P^\Sigma(z) + C_I^\Sigma(z) + C_D^\Sigma(z) \end{aligned}$$

Donde

$$\begin{aligned} C_P^\Sigma(z) &= E(z)K_P \\ C_I^\Sigma(z) &= E(z)K_I \frac{Tz + 1}{2z - 1} \\ C_D^\Sigma(z) &= E(z)K_D \frac{z - 1}{zT} \end{aligned}$$

son las salidas de cada rama. Por tanto, pasando a ecuaciones en diferencias y despejando, obtenemos:

$$\begin{aligned} C_{P_k}^\Sigma &= K_P E_k \\ C_{I_k}^\Sigma &= C_{I_{k-1}}^\Sigma + K_I \frac{T}{2} (E_k + E_{k-1}) \\ C_{D_k}^\Sigma &= K_D \frac{1}{T} (E_k - E_{k-1}) \end{aligned}$$

Siendo la salida del controlador la suma de estas tres últimas expresiones. No obstante, suele ser conveniente hacer unas pequeñas mejoras al diagrama de bloques anterior [9]:

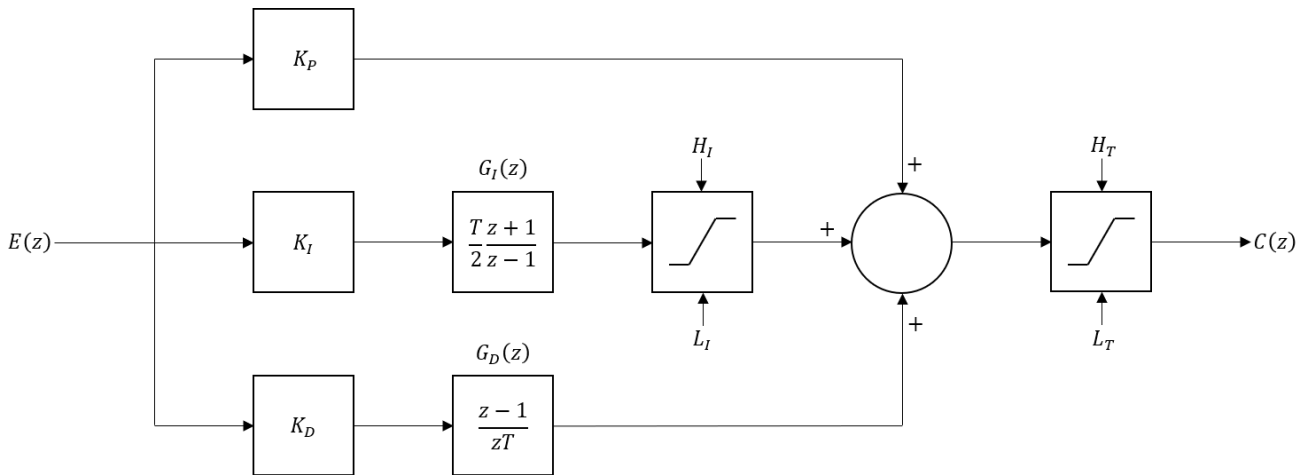


Figura 26 - Controlador PID digital en forma paralela mejorado

A la salida del camino de integración, añadimos un limitador para impedir que la integral acapare la salida del controlador cuando se van acumulando errores muy grandes. Esto se conoce como corrección *anti-windup*. Además, también añadimos otro limitador a la salida. Estos dos limitadores trabajan en conjunto. El limitador a la salida del controlador funciona por saturación estática, es decir, debemos fijar previamente en él el valor mínimo ( $L_T$ ) y máximo ( $H_T$ ) de salida. En base a estos dos valores, y a las salidas del término proporcional ( $P_{out}$ ) y derivativo ( $D_{out}$ ), se calcula el valor mínimo y el máximo del primer limitador:

$$H_I = \max\{H_T - (P_{out} + D_{out}), 0\}$$

$$L_I = \min\{L_T - (P_{out} + D_{out}), 0\}$$

Básicamente estamos limitando el valor del término integral al margen entre el límite exterior y la suma de  $P_{out}$  y  $D_{out}$ . A continuación, una vez revisado el mezclador, explicaremos el cálculo de  $L_T$  y  $H_T$ .

#### 5.3.1.4.2 Mezclador

El mezclador fusiona la referencia de *throttle* y las salidas de los controladores PID para calcular los anchos de pulso de los ESCs.

Como hemos visto anteriormente, los motores 2 y 3 contribuyen positivamente al *roll* y negativamente los motores 1 y 4. Los motores 3 y 4 contribuyen positivamente al *pitch* y negativamente los motores 1 y 2. En cuanto al *yaw rate*, los motores 2 y 4 contribuyen positivamente y los motores 1 y 3 contribuyen negativamente.

Una estrategia para mezclar la referencia de *throttle* con las salidas de los controladores PID, por ejemplo, en el caso del *roll* es:

1. Asignar a todos los ESCs el valor del *throttle*.
2. Sumar al ancho de pulso de los motores 2 y 3 el valor de la salida del controlador PID de *roll*.
3. Restar al ancho de pulso de los motores 1 y 4 el valor de la salida del controlador PID de *roll*.
4. Saturar los anchos de pulso al margen con el que trabajen los ESCs.

Para el resto de los casos la estrategia a seguir es la misma. Haciendo este proceso para el *roll*, para el *pitch* y para el *yaw rate*, llegamos a las siguientes expresiones (antes de limitar los valores):

$$\begin{aligned}\gamma_1 &= throttle - c_\phi - c_\theta - c_r \\ \gamma_2 &= throttle + c_\phi - c_\theta + c_r \\ \gamma_3 &= throttle + c_\phi + c_\theta - c_r \\ \gamma_4 &= throttle - c_\phi + c_\theta + c_r\end{aligned}$$

De cara a la implementación sería suficiente con implementar estos cálculos y luego limitar el resultado al rango en el que trabajen los ESCs. Para el cálculo de  $L_T$  y  $H_T$  es útil expresar estos cálculos en forma matricial:

$$\begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \end{bmatrix} = \begin{bmatrix} +1 & -1 & -1 & -1 \\ +1 & +1 & -1 & +1 \\ +1 & +1 & +1 & -1 \\ +1 & -1 & +1 & +1 \end{bmatrix} \begin{bmatrix} throttle \\ c_\phi \\ c_\theta \\ c_r \end{bmatrix}$$

Si multiplicamos ahora por la inversa de la matriz de signos en ambos miembros de la igualdad, obtenemos:

$$\begin{bmatrix} throttle \\ c_\phi \\ c_\theta \\ c_r \end{bmatrix} = \begin{bmatrix} +1 & -1 & -1 & -1 \\ +1 & +1 & -1 & +1 \\ +1 & +1 & +1 & -1 \\ +1 & -1 & +1 & +1 \end{bmatrix}^{-1} \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{4} & \frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ -\frac{1}{4} & \frac{1}{4} & \frac{1}{4} & -\frac{1}{4} \\ \frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ -\frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & \frac{1}{4} \end{bmatrix} \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \\ \gamma_4 \end{bmatrix}$$

Sustituyendo en esta última ecuación  $\gamma_1 = \gamma_4 = 1000$ ,  $\gamma_2 = \gamma_3 = 2000$ , siendo estos los valores que maximizan la cantidad de *roll* aplicada, obtenemos:

$$\begin{bmatrix} throttle \\ c_\phi \\ c_\theta \\ c_r \end{bmatrix} = \begin{bmatrix} 1500 \\ 500 \\ 0 \\ 0 \end{bmatrix}$$

Por tanto, el valor máximo de salida para el PID de *roll* sería 500. De la misma manera, sustituyendo en la ecuación  $\gamma_1 = \gamma_4 = 2000$  y  $\gamma_2 = \gamma_3 = 1000$ , siendo estos valores los que minimizan la cantidad de *roll* aplicada, obtenemos:

$$\begin{bmatrix} throttle \\ c_\phi \\ c_\theta \\ c_r \end{bmatrix} = \begin{bmatrix} 1500 \\ -500 \\ 0 \\ 0 \end{bmatrix}$$

Se puede ver que el valor máximo que el PID de *roll* puede entregar es de 500 y el mínimo -500 cuando el *throttle* es igual a 1500. De manera intuitiva, esto se debe a que cuando el *throttle* es igual a 1500, el controlador de vuelo tiene el margen máximo de operación sobre los ESCs, pudiendo sumar o restar 500 al valor de cualquier ESC en base a este *throttle* sin entrar en saturación.



Por lo tanto, en el PID de *roll*, podemos fijar los valores de  $H_T$  y  $L_T$  a 500 y  $-500$  respectivamente. Haciendo este mismo procedimiento para el *pitch* y el *yaw rate*, llegamos a la conclusión de que 500 y -500 son también los valores adecuados para el  $H_T$  y  $L_T$  de estos controladores.

### 5.3.2 HRT-HOOD

La descomposición funcional sugiere un diagrama de componentes como el mostrado en la siguiente figura:

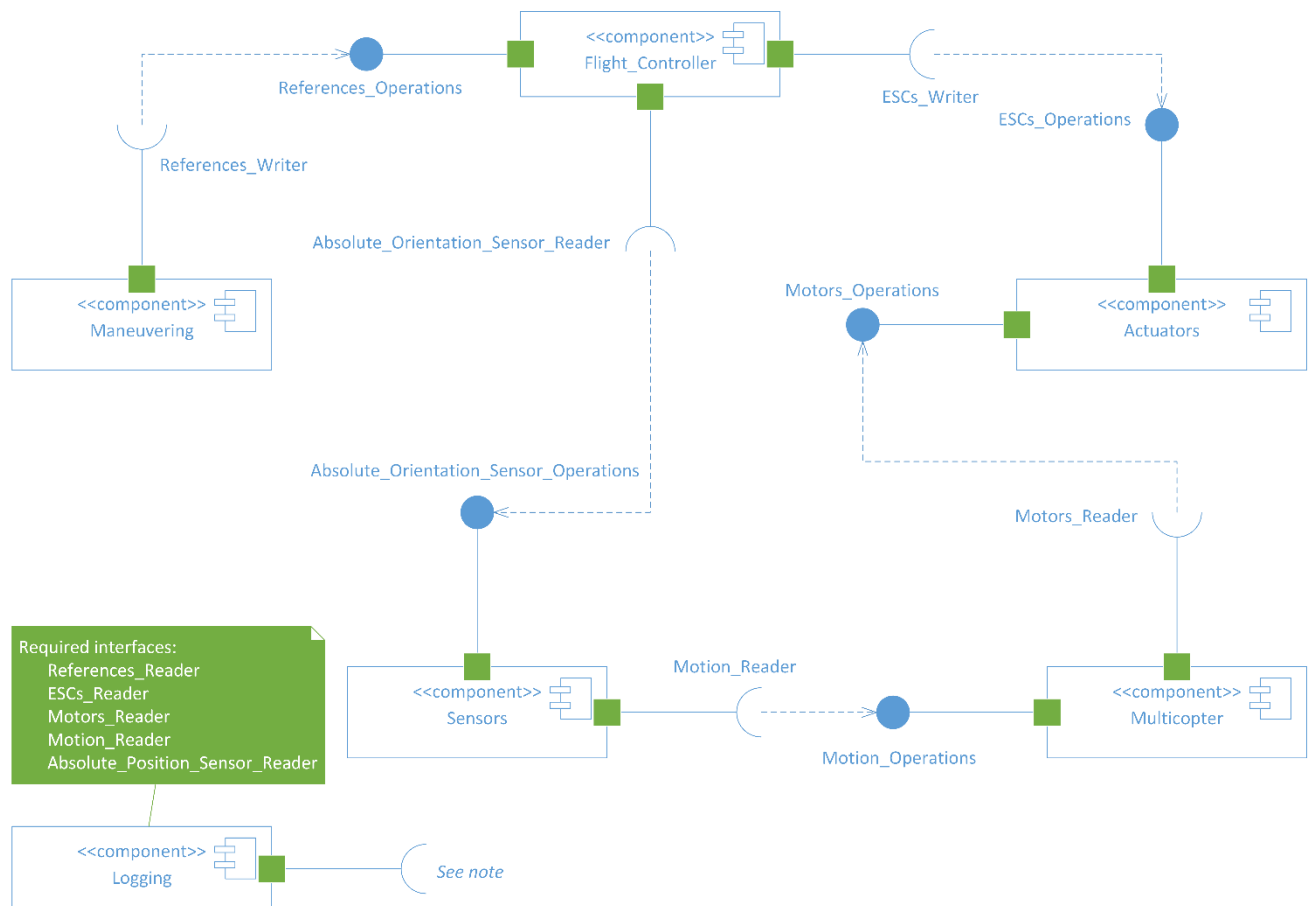


Figura 27 - Diagrama de componentes

En el diagrama, los rectángulos representan los componentes del sistema, los círculos sólidos representan interfaces dispuestas por los componentes, mientras que los semicírculos representan interfaces usadas por los componentes.

Como se puede apreciar, los 4 bloques funcionales del apartado anterior se transforman en componentes en este diagrama. Además, se incluye el componente *Maneuvering*, que servirá para establecer las referencias de vuelo y el componente *Logging*, que servirá, como se ha mencionado antes, para escribir los datos de la simulación en un fichero.

### 5.3.2.1 Maniobras

El componente **Maneuvering** está formado por la tarea periódica *Driver*, que sirve para establecer la referencia de vuelo deseada en cada momento de la simulación. Esta tarea usa la interfaz *References\_Writer* para escribir las referencias en el controlador de vuelo.

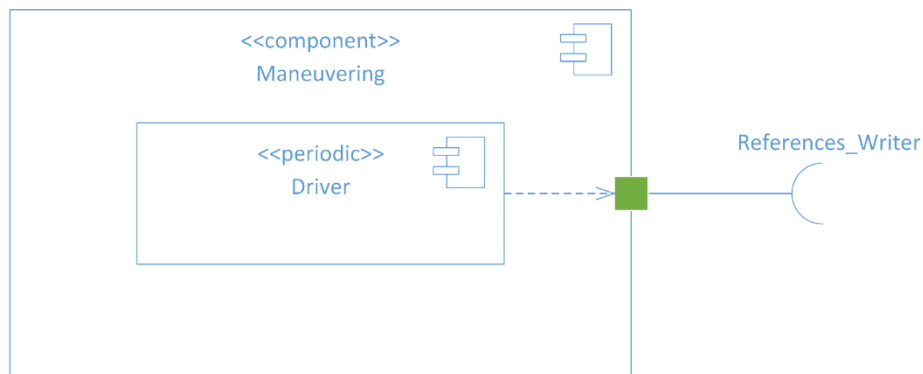


Figura 28 - Componente Maneuvering

### 5.3.2.2 Controlador de vuelo

El componente **Flight\_Controller** está formado por el objeto protegido *References* y por la tarea periódica *Control\_Algorithm*. Como se puede apreciar, el objeto protegido *References* exporta la interfaz *References\_Operations* mientras que la tarea periódica *Control\_Algorithm* se encarga de escribir los anchos de pulso de los ESCs (interfaz *ESCs\_Writer*) a partir de la información de los sensores (interfaz *Absolute\_Orientation\_Sensor\_Reader*) y del objeto protegido anteriormente mencionado.

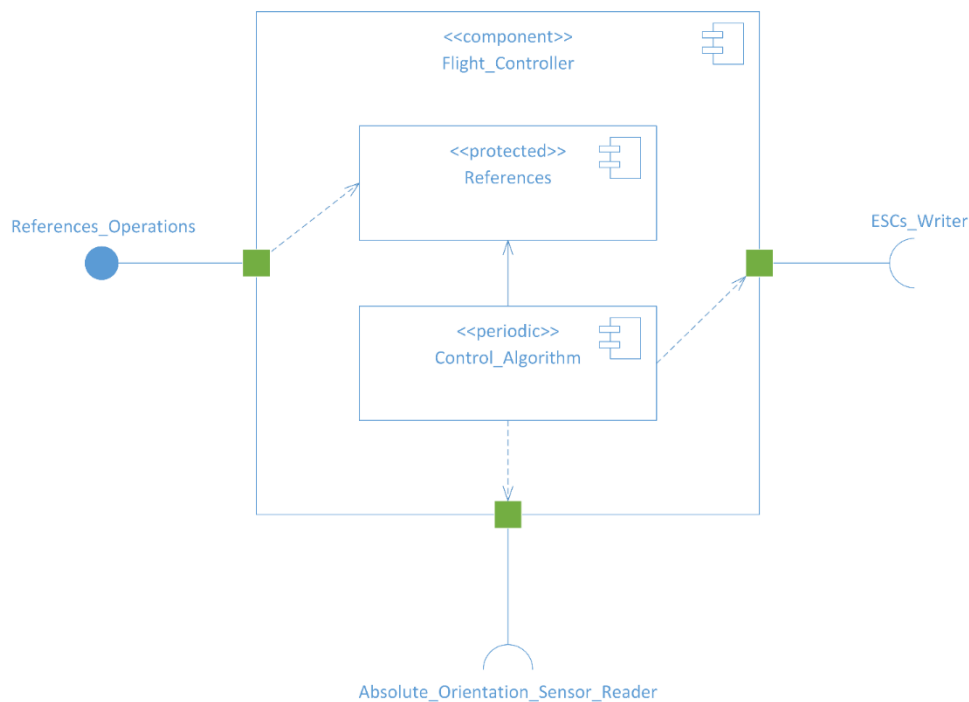


Figura 29 - Componente Flight\_Controller

La interfaz **References\_Operations** se compone a partir de otras 2 interfaces relacionadas; *References\_Reader* y *References\_Writer*. Esta composición sirve para permitir diferentes vistas sobre la interfaz principal, *References\_Operations*. Por un lado, *References\_Reader* sirve para leer las referencias de vuelo mientras que *References\_Writer* sirve para escribir éstas. *References\_Operations* hereda la funcionalidad de estas 2 interfaces.

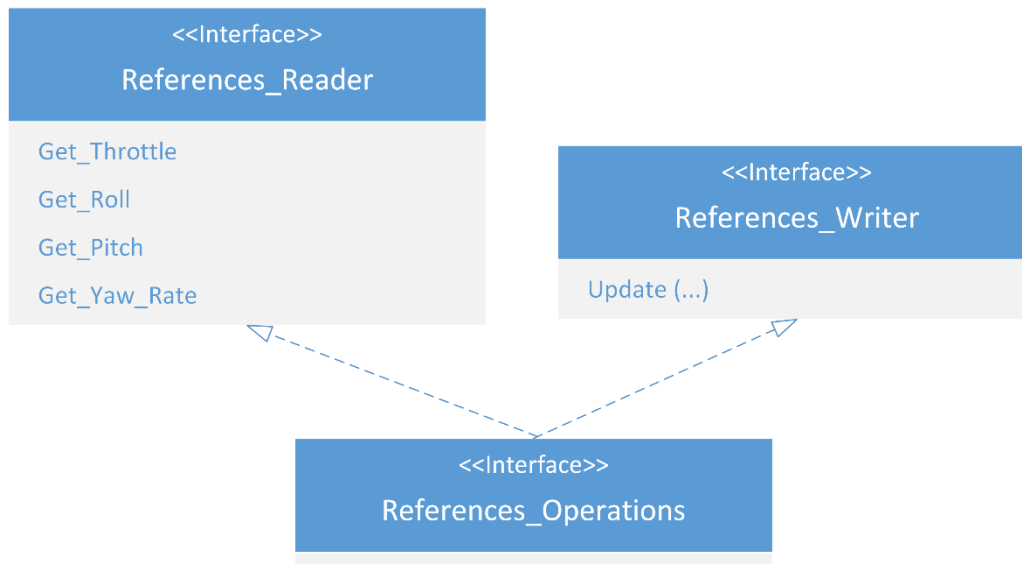


Figura 30 – Interfaces *References\_Reader*, *References\_Writer* y *References\_Operations*

### 5.3.2.3 Actuadores

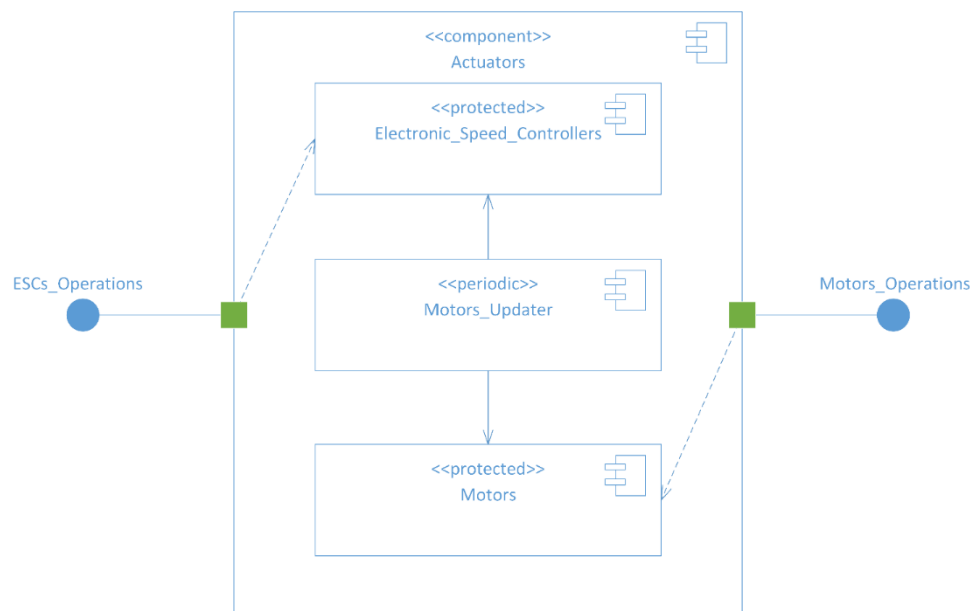


Figura 31 – Componente Actuators

El componente **Actuators** está formado por 2 objetos protegidos y una tarea periódica. El objeto protegido *Electronic\_Speed\_Controllers* exporta la interfaz *ESCs\_Operations* mientras que el objeto protegido *Motors* exporta la interfaz *Motors\_Operations*. La tarea periódica *Motors\_Updater* se encarga de actualizar las velocidades angulares de los motores en base a los anchos de pulsos de los ESCs.

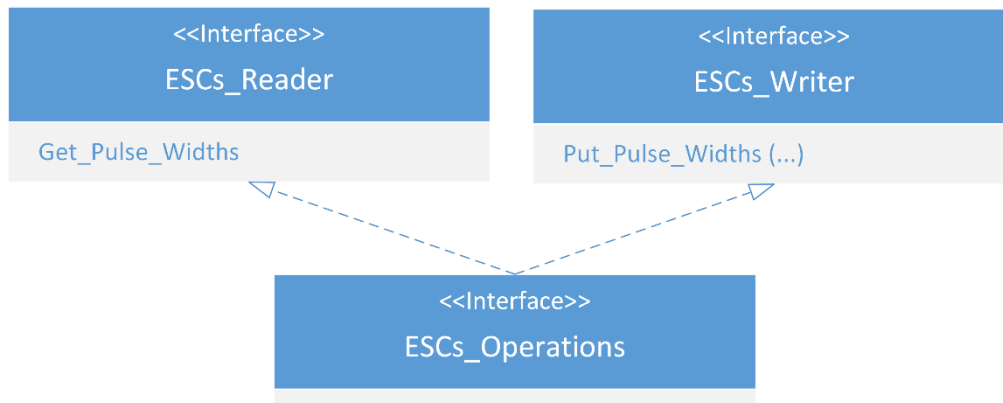


Figura 32 – Interfaces *ESCs\_Reader*, *ESCs\_Writer* y *ESCs\_Operations*

La interfaz **ESCs\_Operations** se compone a partir de otras 2 interfaces relacionadas; *ESCs\_Reader* y *ESCs\_Writer*. Esta composición sirve para permitir diferentes vistas sobre la interfaz principal, *ESCs\_Operations*. Por un lado, *ESCs\_Reader* sirve para leer los anchos de pulso de los ESCs mientras que *ESCs\_Writer* sirve para escribir éstos. *ESCs\_Operations* hereda la funcionalidad de estas dos interfaces.

Por otro lado, la interfaz **Motors\_Operations** se compone a partir de otra interfaz; *Motors\_Reader*. Esta composición sirve para permitir diferentes vistas sobre la interfaz principal, *Motors\_Operations*. *Motors\_Reader* sirve para leer las velocidades angulares de los motores. *Motors\_Operations*, además de heredar la funcionalidad de la interfaz anterior, también provee las funciones para actualizar la referencia de velocidad angular y la velocidad angular real de los motores.

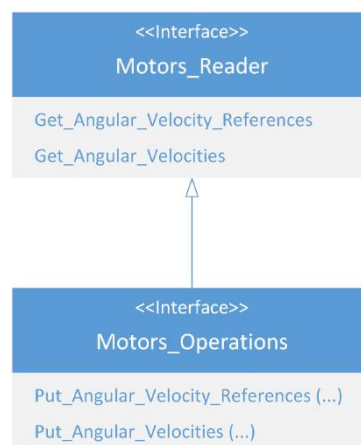


Figura 33 - Interfaces *Motors\_Reader* y *Motors\_Operations*

#### 5.3.2.4 Multicóptero

El componente **Multicopter** está formado por el objeto protegido *Motion*, que exporta la interfaz *Motion\_Operations*, y por la tarea periódica *Motion\_Updater*, que se encarga de actualizar las componentes de movimiento a partir de las velocidades angulares de los motores (interfaz *Motors\_Reader*) y de las componentes de movimiento previas.

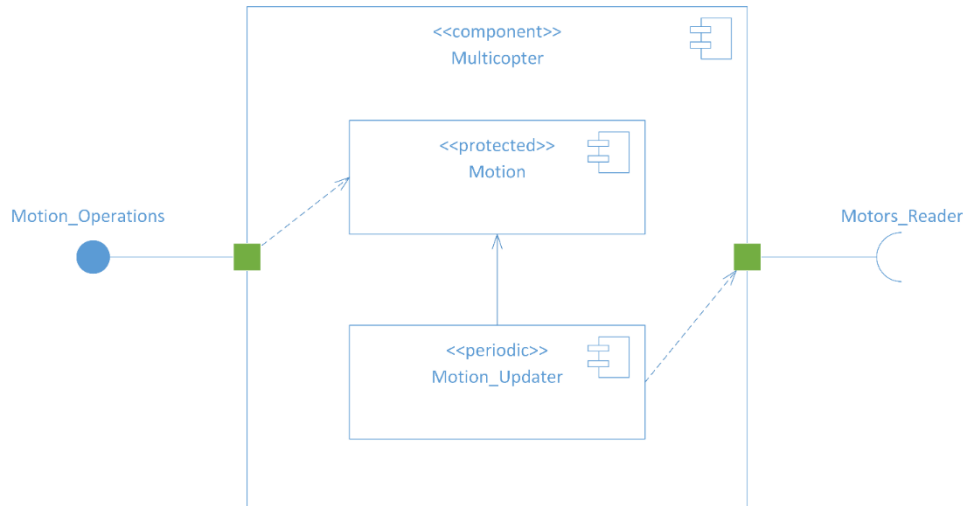


Figura 34 - Componente Multicopter

La interfaz **Motion\_Operations** se compone a partir de otra interfaz; *Motion\_Reader*. Esta composición sirve para permitir diferentes vistas sobre la interfaz principal, *Motion\_Operations*. *Motion\_Reader* sirve para leer las componentes de movimiento, mientras que la interfaz *Motion\_Operations* sirve para escribir éstas y para calcular valores complementarios.

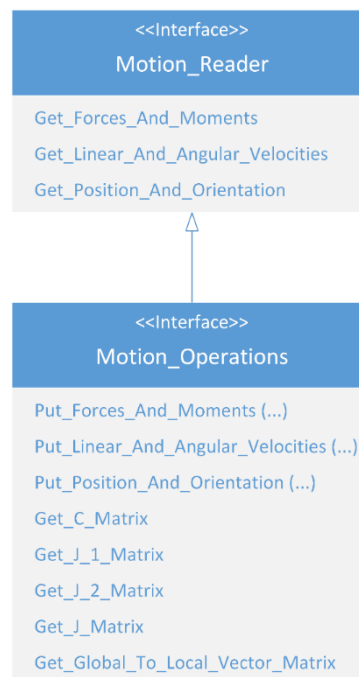


Figura 35 - Interfaces Motion\_Reader y Motion\_Operations

### 5.3.2.5 Sensores

El componente **Sensors** está formado por el objeto protegido *Absolute\_Orientation\_Sensor*, que exporta la interfaz *Absolute\_Orientation\_Sensor\_Operations*, y por la tarea periódica *Absolute\_Orientation\_Sensor\_Updater*, que se encarga de actualizar las componentes de movimiento de este sensor (interfaz *Motion\_Reader*).

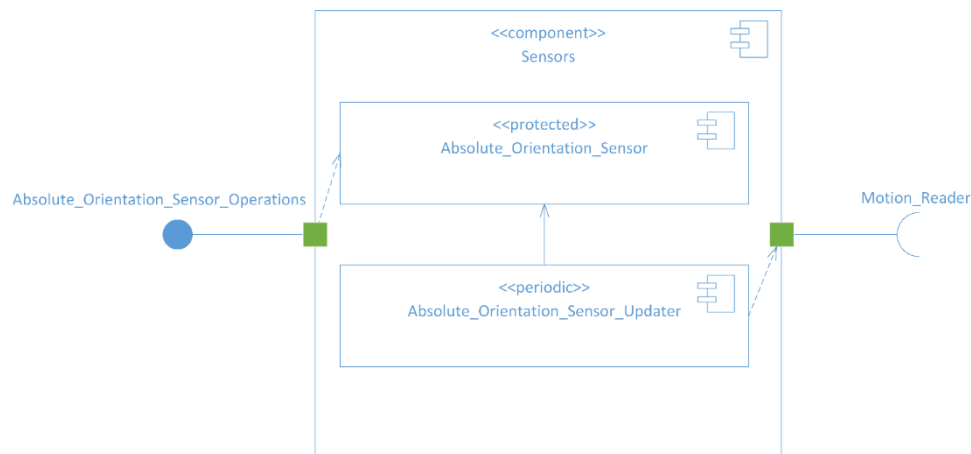


Figura 36 - Componente Sensors

La interfaz ***Absolute\_Orientation\_Sensor\_Operations*** se compone a través de otra interfaz; *Absolute\_Orientation\_Sensor\_Reader*. Esta composición sirve para permitir diferentes vistas sobre la interfaz principal, *Absolute\_Orientation\_Sensor\_Operations*. *Absolute\_Orientation\_Sensor\_Reader* sirve para leer los datos del sensor, mientras que la interfaz *Absolute\_Orientation\_Sensor\_Operations* sirve para escribir éstas.

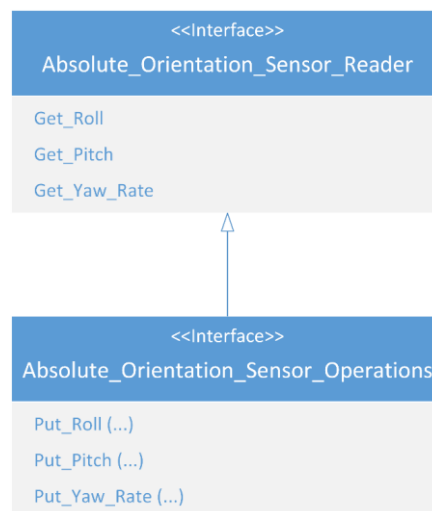


Figura 37 – Interfaces *Absolute\_Orientation\_Sensor\_Reader* y *Absolute\_Orientation\_Sensor\_Operations*

### 5.3.2.6 Logging

El componente **Logging** está formado por la tarea periódica *Logger*, que se encarga de crear y de actualizar el fichero con los datos relevantes de la simulación. Esta tarea hace uso de todas las interfaces de lectura de los demás componentes.

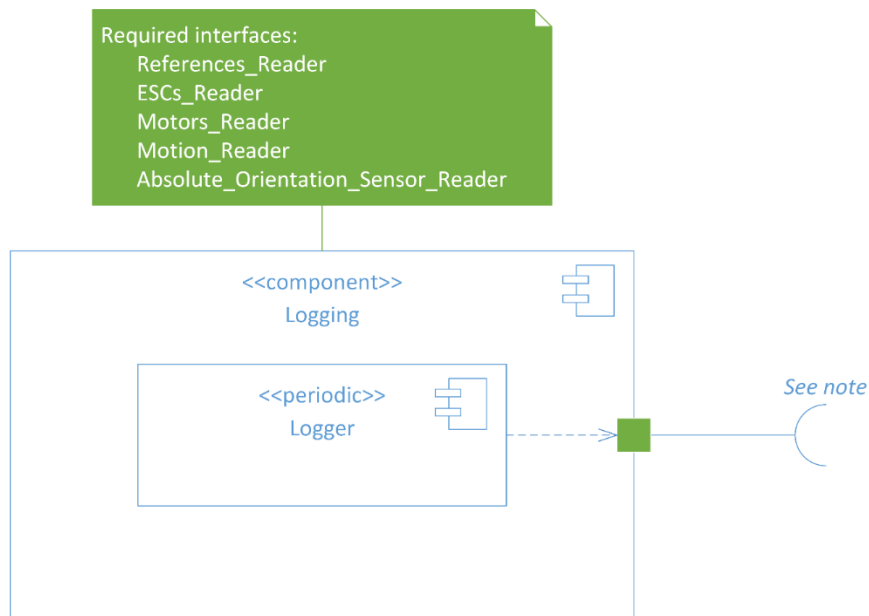


Figura 38 - Componente Logging

## 5.4 Implementación

Para la implementación en Ada, creamos un paquete por cada componente que aparezca en el diagrama, es decir, se crean los paquetes *Maneuvering*, *Flight\_Controller*, *Actuators*, *Sensors* y *Logging*. El siguiente paso es crear, para cada paquete anterior, sub-paquetes que representen los objetos protegidos o tareas periódicas dentro del componente en cuestión. Por ejemplo, para el caso del paquete *Actuators*, creamos los sub-paquetes *Electronic\_Speed\_Controllers*, *Motors* y *Motors\_Updater*.

Cada paquete del nivel superior define al menos las interfaces exportadas por el componente al que representa, mientras que los sub-paquetes que contiene implementarán esas interfaces o tareas periódicas.

Para simplificar la implementación de algunos paquetes, se puede extraer fuera de éstos aquellas funcionalidades que se puedan reutilizar (por ejemplo, la implementación de los controladores PID) y/o que sean susceptibles de representar una entidad por sí mismas (por ejemplo, el mezclador).

La estructura de la implementación se basa en el ejemplo presentado en [1].



### 5.4.1 Estructura de archivos

La estructura de archivos es la siguiente:

```

├── examples
│   └── basic_example
│       ├── basic_example.gpr
│       ├── obj
│       └── src
│           ├── configuration.ads
│           ├── driver.adb
│           ├── driver.ads
│           └── main.adb
├── LICENSE
├── README.md
└── simulator
    ├── lib
    ├── obj
    ├── simulator.gpr
    └── src
        ├── actuators.ads
        ├── actuators-electronic_speed_controllers.adb
        ├── actuators-electronic_speed_controllers.ads
        ├── actuators-motors.adb
        ├── actuators-motors.ads
        ├── actuators-motors_updater.adb
        ├── actuators-motors_updater.ads
        ├── cyclics.adb
        ├── cyclics.ads
        ├── flight_controller.ads
        ├── flight_controller-control_algorithm.adb
        ├── flight_controller-control_algorithm.ads
        ├── flight_controller-references.adb
        ├── flight_controller-references.ads
        ├── logging.ads
        ├── logging-logger.adb
        ├── logging-logger.ads
        ├── maneuvering.ads
        ├── maneuvering-driver.ads
        ├── mixers.adb
        ├── mixers.ads
        ├── multicopter.ads
        ├── multicopter-motion.adb
        ├── multicopter-motion.ads
        ├── multicopter-motion_updater.adb
        ├── multicopter-motion_updater.ads
        ├── pid_controllers.adb
        ├── pid_controllers.ads
        ├── sensors-absolute_orientation_sensor.adb
        ├── sensors-absolute_orientation_sensor.ads
        ├── sensors-absolute_orientation_sensor_updater.adb
        ├── sensors-absolute_orientation_sensor_updater.ads
        ├── sensors.ads
        ├── utils.adb
        └── utils.ads

```

### 5.4.2 Útiles

Para evitar la repetición innecesaria de código, usamos una plantilla para las tareas periódicas. La especificación e implementación de esta plantilla se encuentra, respectivamente, en los ficheros *cyclics.ads* y *cyclics.adb*:

```
-- cyclics.ads

with Ada.Real_Time; use Ada.Real_Time;

with System; use System;

package Cyclics is

    type Bundle is abstract tagged limited
        record
            Task_Priority : Priority;
            Release_Interval : Time_Span;
            Task_Duration : Time_Span;
        end record;

    procedure On_Start (Self : in out Bundle) is abstract;
    procedure On_Loop (Self : in out Bundle; Elapsed_Time : Time_Span) is abstract;
    procedure On_End (Self : in out Bundle) is abstract;
    type Any_Bundle is access all Bundle'Class;
    task type Cyclic (Bundle : Any_Bundle) is
        pragma Priority (Bundle.Task_Priority);
    end Cyclic;
end Cyclics;
```

```

-- cyclics.adb

package body Cyclics is

    -----
    -- Cyclic --
    -----

    task body Cyclic is

        Next_Release : Time;

        Start_Time : Time;

    begin

        Start_Time := Clock;

        Bundle.On_Start;

        Next_Release := Clock + Bundle.Release Interval;

    loop

        Bundle.On_Loop (Clock - Start_Time);

        delay until Next_Release;

        Next_Release := Next_Release + Bundle.Release_Interval;

        exit when Clock - Start_Time >= Bundle.Task_Duration;

    end loop;

    Bundle.On_End;

end Cyclic;

end Cyclics;

```

También creamos el paquete *Utils*, que contiene funciones usadas en varios paquetes del proyecto o funciones de utilidad. La especificación e implementación de paquete se encuentra, respectivamente, en los ficheros *utils.ads* y *utils.adb*:

```

-- utils.ads

with Ada.Numerics.Elementary_Functions; use Ada.Numerics.Elementary_Functions;
with Ada.Numerics.Real_Arrays; use Ada.Numerics.Real_Arrays;

package Utils is

    function Clamp (N : Float; Min : Float; Max : Float) return Float;

    function Clamp (N : Real_Vector; Min : Float; Max : Float) return Real_Vector;

    function Linear_Interpolation (X, X_0, X_1, Y_0, Y_1 : Float) return Float;

    function Degrees_To_Radians (Degrees : Float) return Float;

end Utils;

```

```

-- utils.adb

package body Utils is

  -----
  -- Clamp --
  -----

  function Clamp (N : Float; Min : Float; Max: Float) return Float is
  begin

    if N > Max then

      return Max;

    elsif N < Min then

      return Min;

    else

      return N;

    end if;

  end Clamp;

  -----
  -- Clamp --
  -----

  function Clamp (N : Real Vector; Min : Float; Max : Float) return Real Vector is

    N_Clamped : Real_Vector (N'First .. N'Last);

  begin

    for I in N_Clamped'Range loop

      N_Clamped (I) := Clamp (N (I), Min, Max);

    end loop;

    return N_Clamped;

  end Clamp;

  -----
  -- Linear_Interpolation --
  -----

  function Linear_Interpolation (X, X_0, X_1, Y_0, Y_1 : Float) return Float is
  begin
    return (Y_0 * (X_1 - X) + Y_1 * (X - X_0)) / (X_1 - X_0);
  end Linear_Interpolation;

  -----
  -- Degrees_To_Radians --
  -----

  function Degrees_To_Radians (Degrees : Float) return Float is
  begin
    return Degrees * (Ada.Numerics.Pi / 180.0);
  end Degrees_To_Radians;

end Utils;

```

### 5.4.3 Maniobras

El paquete *Maneuvering* no define ninguna interfaz, por lo que simplemente importamos aquellos paquetes de los que dependa. Como convenio, y de ahora en adelante, usamos la cláusula *with* con su correspondiente cláusula *use* cuando utilizamos paquetes que no pertenezcan al simulador, y sólo usamos la cláusula *with* (sin la cláusula *use*) cuando utilizamos paquetes que pertenezcan al simulador. La especificación del paquete *Maneuvering* es:

```
-- maneuvering.ads

with Ada.Real_Time; use Ada.Real_Time;

with Cyclics;
with Flight_Controller;

package Maneuvering is

end Maneuvering;
```

Para la tarea periódica *Driver*, especificamos sólo el esqueleto, ya que la implementación dependerá de las referencias de vuelo que se quieran usar. La especificación es, por tanto:

```
-- maneuvering-driver.ads

package Maneuvering.Driver is

  type Bundle (References : Flight_Controller.Any_References_Writer) is abstract new Cyclics.Bundle
  with null record;

  procedure On_Start (Self : in out Bundle) is abstract;

  procedure On_Loop (Self : in out Bundle; Elapsed_Time : Time_Span) is abstract;

  procedure On_End (Self : in out Bundle) is abstract;

end Maneuvering.Driver;
```

Como ejemplo de uso, podemos recurrir al código de la primera prueba que veremos en el siguiente apartado:

```
-- driver.ads

with Ada.Real_Time; use Ada.Real_Time;

with Maneuvering.Driver;
with Utils;

package Driver is

  type Bundle is new Maneuvering.Driver.Bundle with null record;

  procedure On_Start (Self : in out Bundle);

  procedure On_Loop (Self : in out Bundle; Elapsed_Time : Time_Span);

  procedure On_End (Self : in out Bundle);

end Driver;
```

```

-- driver.adb

package body Driver is

    -----
    -- On_Start --
    -----

    procedure On_Start (Self : in out Bundle) is

        Throttle_Reference : constant Float := 1000.0;
        Roll_Reference     : constant Float := Utils.Degrees_To_Radians (0.0);
        Pitch_Reference     : constant Float := Utils.Degrees_To_Radians (0.0);
        Yaw_Rate_Reference  : constant Float := Utils.Degrees_To_Radians (0.0);

    begin

        Self.References.Update (Throttle_Reference, Roll_Reference, Pitch_Reference,
        Yaw_Rate_Reference);

    end On_Start;

    -----
    -- On_Loop --
    -----

    procedure On_Loop (Self : in out Bundle; Elapsed_Time : Time_Span) is
    begin
        null;
    end On_Loop;

    -----
    -- On_End --
    -----

    procedure On_End (Self : in out Bundle) is
    begin
        null;
    end On_End;

end Driver;

```

#### 5.4.4 Controlador de vuelo

Antes de ver la especificación e implementación del paquete *Flight\_Controller* y de sus sub-paquetes, veremos el contenido de los paquetes *PID\_Controllers* y *Mixers*, pues como se ha comentado antes, estas dos funcionalidades se han extraído del paquete *Flight\_Controller*. La especificación e implementación del paquete *PID\_Controllers* es:

```
-- pid_controllers.ads

with Utils;

package PID_Controllers is

  Illegal_Sample_Time : Exception;
  Illegal_Output_Limits : Exception;

  type Parallel_PID_Controller is tagged private;

  type Parallel_PID_Controller_Access is access all Parallel_PID_Controller'Class;

  function Create (KP : Float; KI : Float; KD : Float; Sample_Time : Float; Min_Output : Float;
                  Max_Output : Float) return Parallel_PID_Controller;

  function Update (Self : in out Parallel_PID_Controller; Reference : Float; Measured_Value : Float)
return Float;

  procedure Reset (Self : in out Parallel_PID_Controller);

private

  type Parallel_PID_Controller is tagged

    record

      KP : Float;
      KI : Float;
      KD : Float;

      Sample_Time : Float;

      Min_Output : Float;
      Max_Output : Float;

      Past_Error : Float;

      Past_Integral_Output : Float;

    end record;

end PID_Controllers;
```

```

-- pid_controllers.adb

package body PID_Controllers is

    -----
    -- Create --
    -----

    function Create (KP : Float; KI : Float; KD : Float; Sample_Time : Float; Min_Output : Float;
                    Max_Output : Float) return Parallel_PID_Controller is
    begin
        -- Validate Sample_Time.

        if Sample_Time = 0.0 then
            raise Illegal_Sample_Time with "Sample time must be greater than 0";
        end if;

        -- Validate Min Output and Max Output.

        if not (Min_Output < Max_Output) then
            raise Illegal_Output_Limits with "Min_Output must be lower than Max_Output";
        end if;

        -- Create and return.

        return (KP => KP, KI => KI, KD => KD, Sample_Time => Sample_Time, Min_Output => Min_Output,
                Max_Output => Max_Output, Past_Error => 0.0, Past_Integral_Output => 0.0);
    end Create;

    -----
    -- Update --
    -----

    function Update (Self : in out Parallel_PID_Controller; Reference : Float; Measured_Value : Float)
    return Float is

        Alpha : constant Float := (Self.Sample_Time / 2.0) * Self.KI;
        Beta : constant Float := (1.0 / Self.Sample_Time) * Self.KD;

        Error : Float;

        Proportional_Output : Float;
        Integral_Output : Float;
        Derivative_Output : Float;

        Integral_Min_Output : Float;
        Integral_Max_Output : Float;

        Total_Output : Float;

    begin
        -- Compute the error.

        Error := Reference - Measured_Value;

        -- Compute the output for each block.

        Proportional_Output := Self.KP * Error;
        Integral_Output := Self.Past_Integral_Output + Alpha * (Error + Self.Past_Error);
        Derivative_Output := Beta * (Error - Self.Past_Error);

        -- Anti-Windup.

        Integral_Min_Output := Float'Min (Self.Min_Output - (Proportional_Output + Derivative_Output),
0.0);
        Integral_Max_Output := Float'Max (Self.Max_Output - (Proportional_Output + Derivative_Output),
0.0);

        Integral_Output := Utils.Clamp (Integral_Output, Integral_Min_Output, Integral_Max_Output);
    end Update;
end PID_Controllers;

```



```

    -- Compute and limit the total output.

    Total_Output := Proportional_Output + Integral_Output + Derivative_Output;

    Total_Output := Utils.Clamp (Total_Output, Self.Min_Output, Self.Max_Output);

    -- Prepare the next iteration.

    Self.Past_Error := Error;
    Self.Past_Integral_Output := Integral_Output;

    -- Return the total output.

    return Total_Output;

end Update;

-----
-- Reset --
-----

procedure Reset (Self : in out Parallel_PID_Controller) is
begin
    -- Set to zero the memory components.

    Self.Past_Error := 0.0;
    Self.Past_Integral_Output := 0.0;

end Reset;

end PID_Controllers;

```

La especificación e implementación del paquete *Mixers* es:

```

-- mixers.ads

with Ada.Numerics.Real_Arrays; use Ada.Numerics.Real_Arrays;

with Utils;

package Mixers is

    Illegal_Output_Limits : Exception;

    type Quadcopter_Cross_Mixer is tagged private;

    type Quadcopter_Cross_Mixer_Access is access all Quadcopter_Cross_Mixer'Class;

    function Create (ESC_Min_Pulse_Width : Float; ESC_Max_Pulse_Width : Float) return
    Quadcopter_Cross_Mixer;

    function Mix (Self : in out Quadcopter_Cross_Mixer; Throttle : Float; Roll_PID_Output : Float;
    Pitch_PID_Output : Float; Yaw_Rate_PID_Output : Float) return Real_Vector;

private

    type Quadcopter_Cross_Mixer is tagged

        record

            ESC_Min_Pulse_Width : Float;
            ESC_Max_Pulse_Width : Float;

        end record;

end Mixers;

```

```

-- mixers.adb

package body Mixers is

    -----
    -- Create --
    -----

    function Create (ESC_Min_Pulse_Width : Float; ESC_Max_Pulse_Width : Float) return
    Quadcopter_Cross_Mixer is
    begin
        -- Validate ESC_Min_Pulse_Width and ESC_Max_Pulse_Width.

        if not (ESC_Min_Pulse_Width < ESC_Max_Pulse_Width) then

            raise Illegal_Output_Limits with "ESC_Min_Pulse_Width must be lower than
            ESC_Max_Pulse_Width";

        end if;

        -- Create and return.

        return (ESC_Min_Pulse_Width => ESC_Min_Pulse_Width, ESC_Max_Pulse_Width =>
        ESC_Max_Pulse_Width);

    end Create;

    -----
    -- Mix --
    -----

    function Mix (Self : in out Quadcopter_Cross_Mixer; Throttle : Float; Roll_PID_Output : Float;
    Pitch_PID_Output : Float; Yaw_Rate_PID_Output : Float) return Real_Vector is

        X : Float renames Roll_PID_Output;
        Y : Float renames Pitch_PID_Output;
        Z : Float renames Yaw_Rate_PID_Output;

        Mix : Real_Vector (1 .. 4);

    begin

        Mix (1) := Utils.Clamp (Throttle - X - Y - Z, Self.ESC_Min_Pulse_Width,
        Self.ESC_Max_Pulse_Width);
        Mix (2) := Utils.Clamp (Throttle + X - Y + Z, Self.ESC_Min_Pulse_Width,
        Self.ESC_Max_Pulse_Width);
        Mix (3) := Utils.Clamp (Throttle + X + Y - Z, Self.ESC_Min_Pulse_Width,
        Self.ESC_Max_Pulse_Width);
        Mix (4) := Utils.Clamp (Throttle - X + Y + Z, Self.ESC_Min_Pulse_Width,
        Self.ESC_Max_Pulse_Width);

        return Mix;

    end Mix;

end Mixers;

```

El paquete *Flight\_Controller* define las interfaces *References\_Reader*, *References\_Writer* y *References\_Operations*. Además, también se especifican en él algunas constantes:

```
-- flight_controller.ads

with Ada.Numerics.Real_Arrays; use Ada.Numerics.Real_Arrays;
with Ada.Real_Time; use Ada.Real_Time;

with System; use System;

with Actuators;
with Cyclics;
with Mixers;
with PID_Controllers;
with Sensors;
with Utils;

package Flight_Controller is

    -----
    -- Constants --
    -----

    Min_Throttle : constant Float := 1000.0;

    Max_Throttle : constant Float := 2000.0;

    Min_Roll : constant Float := Utils.Degrees_To_Radians (-25.0);
    Max_Roll : constant Float := Utils.Degrees_To_Radians (+25.0);
    Min_Pitch : constant Float := Utils.Degrees_To_Radians (-25.0);
    Max_Pitch : constant Float := Utils.Degrees_To_Radians (+25.0);
    Min_Yaw_Rate : constant Float := Utils.Degrees_To_Radians (-180.0);
    Max_Yaw_Rate : constant Float := Utils.Degrees_To_Radians (+180.0);

    -----
    -- References --
    -----

    -- References_Reader interface.

    type References_Reader is synchronized interface;

    function Get_Throttle (Self: References_Reader) return Float is abstract;

    function Get_Roll (Self: References_Reader) return Float is abstract;

    function Get_Pitch (Self: References_Reader) return Float is abstract;

    function Get_Yaw_Rate (Self: References_Reader) return Float is abstract;

    type Any_References_Reader is access all References_Reader'Class;

    -- References_Writer interface.

    type References_Writer is synchronized interface;

    procedure Update (Self: in out References_Writer; Throttle, Roll, Pitch, Yaw_Rate : Float) is
    abstract;

    type Any_References_Writer is access all References_Writer'Class;

    -- References_Operations interface.

    type References_Operations is synchronized interface and References_Reader and References_Writer;

    type Any_References_Operations is access all References_Operations'Class;

end Flight_Controller;
```

El paquete *Flight\_Controller.References* implementa la interfaz *References\_Operations* usando un objeto protegido:

```
-- flight_controller-references.ads

package Flight_Controller.References is

  protected type Agent (Ceiling: Priority) with Priority => Ceiling is new References_Operations
  with

    overriding function Get_Throttle return Float;

    overriding function Get_Roll return Float;

    overriding function Get_Pitch return Float;

    overriding function Get_Yaw_Rate return Float;

    overriding procedure Update (Throttle : Float; Roll : Float; Pitch : Float; Yaw_Rate : Float);

  private

    Throttle : Float := Flight_Controller.Min_Throttle;

    Roll : Float := 0.0;

    Pitch : Float := 0.0;

    Yaw_Rate : Float := 0.0;

  end Agent;

end Flight_Controller.References;

-- flight_controller-references.adb

package body Flight_Controller.References is

  -----
  -- Agent --
  -----

  protected body Agent is

    -----
    -- Get_Throttle --
    -----

    function Get_Throttle return Float is
    begin
      return Agent.Throttle;
    end Get_Throttle;

    -----
    -- Get_Roll --
    -----

    function Get_Roll return Float is
    begin
      return Agent.Roll;
    end Get_Roll;

    -----
    -- Get_Pitch --
    -----

    function Get_Pitch return Float is
    begin
      return Agent.Pitch;
    end Get_Pitch;

    -----
    -- Get_Yaw_Rate --
    -----
```

```

function Get_Yaw_Rate return Float is
begin
    return Agent.Yaw_Rate;
end Get_Yaw_Rate;

-----
-- Update --
-----

procedure Update (Throttle : Float; Roll : Float; Pitch : Float; Yaw_Rate : Float) is
begin
    Agent.Throttle := Utils.Clamp (Throttle, Flight_Controller.Min_Throttle,
Flight_Controller.Max_Throttle);
    Agent.Roll := Utils.Clamp (Roll, Flight_Controller.Min_Roll, Flight_Controller.Max_Roll);
    Agent.Pitch := Utils.Clamp (Pitch, Flight_Controller.Min_Pitch,
Flight_Controller.Max_Pitch);
    Agent.Yaw_Rate := Utils.Clamp (Yaw_Rate, Flight_Controller.Min_Yaw_Rate,
Flight_Controller.Max_Yaw_Rate);

    end Update;

end Agent;

end Flight_Controller.References;

```

El paquete *Flight\_Controller.Control\_Algorithm*, especifica e implementa la tarea periódica *Control\_Algorithm*:

```

-- flight controller-control algorithm.ads

package Flight_Controller.Control_Algorithm is

    type Bundle (Roll_PID : PID.Controllers.Parallel_PID_Controller_Access;
Pitch_PID : PID.Controllers.Parallel_PID_Controller_Access;
Yaw_Rate_PID : PID.Controllers.Parallel_PID_Controller_Access;
Quadcopter_Cross_Mixer : Mixers.Quadcopter_Cross_Mixer_Access;
References : Any_References_Reader;
ESCs : Actuators.Any_ESCs_Writer;
Absolute_Orientation_Sensor : Sensors.Any_Absolute_Orientation_Sensor_Reader) is

        new Cyclics.Bundle with null record;

    procedure On_Start (Self : in out Bundle);

    procedure On_Loop (Self : in out Bundle; Elapsed_Time : Time_Span);

    procedure On_End (Self : in out Bundle);

end Flight_Controller.Control_Algorithm;

```

```

-- flight_controller-control_algorithm.adb

package body Flight_Controller.Control_Algorithm is

    -----
    -- On_Start --
    -----

    procedure On_Start (Self : in out Bundle) is
    begin
        null;
    end On_Start;

    -----
    -- On_Loop --
    -----

    procedure On_Loop (Self : in out Bundle; Elapsed_Time : Time_Span) is

        Commanded_Throttle : Float;
        Commanded_Roll : Float;
        Commanded_Pitch : Float;
        Commanded_Yaw_Rate : Float;

        Measured_Roll : Float;
        Measured_Pitch : Float;
        Measured_Yaw_Rate : Float;

        Roll_PID_Output : Float;
        Pitch_PID_Output : Float;
        Yaw_Rate_PID_Output : Float;

        Mix : Real_Vector (1 .. Actuators.Number_Of_Actuators);

    begin

        -- Get commanded values.

        Commanded_Throttle := Self.References.Get_Throttle;
        Commanded_Roll := Self.References.Get_Roll;
        Commanded_Pitch := Self.References.Get_Pitch;
        Commanded_Yaw_Rate := Self.References.Get_Yaw_Rate;

        -- Get measured values.

        Measured_Roll := Self.Absolute_Orientation_Sensor.Get_Roll;
        Measured_Pitch := Self.Absolute_Orientation_Sensor.Get_Pitch;
        Measured_Yaw_Rate := Self.Absolute_Orientation_Sensor.Get_Yaw_Rate;

        -- Compute controller outputs.

        Roll_PID_Output := Self.Roll_PID.Update (Commanded_Roll, Measured_Roll);
        Pitch_PID_Output := Self.Pitch_PID.Update (Commanded_Pitch, Measured_Pitch);
        Yaw_Rate_PID_Output := Self.Yaw_Rate_PID.Update (Commanded_Yaw_Rate, Measured_Yaw_Rate);

        -- Mix throttle with PID outputs.

        Mix := Self.Quadcopter_Cross_Mixer.Mix (Commanded_Throttle, Roll_PID_Output, Pitch_PID_Output,
        Yaw_Rate_PID_Output);

        -- Write ESCs.

        Self.ESCs.Put_Pulse_Widths (Mix);

    end On_Loop;

    -----
    -- On_End --
    -----

    procedure On_End (Self : in out Bundle) is
    begin
        null;
    end On_End;

end Flight_Controller.Control_Algorithm;

```

### 5.4.5 Actuadores

El paquete *Actuators* define las interfaces *ESCs\_Reader*, *ESCs\_Writer* y *ESCs\_Operations*, *Motors\_Reader* y *Motors\_Operations*. Además, también se especifican en él algunas constantes:

```
-- actuators.ads

with Ada.Numerics.Elementary_Functions; use Ada.Numerics.Elementary_Functions;
with Ada.Numerics.Real_Arrays; use Ada.Numerics.Real_Arrays;
with Ada.Real_Time; use Ada.Real_Time;

with System; use System;

with Cyclics;
with Utils;

package Actuators is

    -----
    -- Constants --
    -----

    Number_Of_Actuators : constant Positive := 4;

    ESC_Min_Pulse_Width : constant Float := 1000.0;

    ESC_Max_Pulse_Width : constant Float := 2000.0;

    Motor_Min_Angular_Velocity : constant Float := 0.0;

    Motor_Max_Angular_Velocity : constant Float := 1240.0;

    Motors_Setting_Time : constant Float := 0.250;

    -----
    -- Electronic speed controllers --
    -----

    -- ESCs_Reader interface.

    type ESCs_Reader is synchronized interface;

    function Get_Pulse_Widths (Self : ESCs_Reader) return Real_Vector is abstract;

    type Any_ESCs_Reader is access all ESCs_Reader'Class;

    -- ESCs_Writer interface.

    type ESCs_Writer is synchronized interface;

    procedure Put_Pulse_Widths (Self : in out ESCs_Writer; Pulse_Widths : Real_Vector) is abstract;

    type Any_ESCs_Writer is access all ESCs_Writer'Class;

    -- ESCs_Operations interface.

    type ESCs_Operations is synchronized interface and ESCs_Reader and ESCs_Writer;

    type Any_ESCs_Operations is access all ESCs_Operations'Class;

    -----
    -- Motors --
    -----

    -- Motors_Reader interface.

    type Motors_Reader is synchronized interface;

    function Get_Angular_Velocity_References (Self: Motors_Reader) return Real_Vector is abstract;

    function Get_Angular_Velocities (Self: Motors_Reader) return Real_Vector is abstract;

    type Any_Motors_Reader is access all Motors_Reader'Class;

    -- Motors_Operations interface.
```

```

type Motors_Operations is synchronized interface and Motors_Reader;

procedure Put_Angular_Velocity_References (Self : in out Motors_Operations;
                                           Angular_Velocity_References : Real_Vector) is abstract;

procedure Put_Angular_Velocities (Self : in out Motors_Operations; Angular_Velocities :
Real_Vector) is abstract;

type Any_Motors_Operations is access all Motors_Operations'Class;

end Actuators;

```

El paquete *Actuators.Electronic\_Speed\_Controllers* implementa la interfaz *ESCs\_Operations* usando un objeto protegido:

```

-- actuators-electronic speed controllers.ads

package Actuators.Electronic_Speed_Controllers is

    protected type Agent (Ceiling: Priority) with Priority => Ceiling is new ESCs_Operations with

        overriding function Get_Pulse_Widths return Real_Vector;

        overriding procedure Put_Pulse_Widths (Pulse_Widths : Real_Vector);

    private

        Pulse_Widths : Real_Vector (1 .. Actuators.Number_Of_Actuators) := (others =>
Actuators.ESC_Min_Pulse_Width);

    end Agent;

end Actuators.Electronic_Speed_Controllers;

-- actuators-electronic_speed_controllers.adb

package body Actuators.Electronic_Speed_Controllers is

    -----
    -- Agent --
    -----

    protected body Agent is

        -----
        -- Get_Pulse_Widths --
        -----

        function Get_Pulse_Widths return Real_Vector is
        begin
            return Agent.Pulse_Widths;
        end Get_Pulse_Widths;

        -----
        -- Put_Pulse_Widths --
        -----

        procedure Put_Pulse_Widths (Pulse_Widths : Real_Vector) is
        begin
            Agent.Pulse_Widths := Utils.Clamp (Pulse_Widths, Actuators.ESC_Min_Pulse_Width,
Actuators.ESC_Max_Pulse_Width);
        end Put_Pulse_Widths;

    end Agent;

end Actuators.Electronic_Speed_Controllers;

```



El paquete *Actuators.Motors* implementa la interfaz *Motors\_Operations* usando un objeto protegido:

```
-- actuators-motors.ads

package Actuators.Motors is

  protected type Agent (Ceiling: Priority) with Priority => Ceiling is new Motors_Operations with

    overriding function Get_Angular_Velocity_References return Real_Vector;

    overriding function Get_Angular_Velocities return Real_Vector;

    overriding procedure Put_Angular_Velocity_References (Angular_Velocity_References :
Real_Vector);

    overriding procedure Put_Angular_Velocities (Angular_Velocities : Real_Vector);

  private

    Angular_Velocity_References : Real_Vector (1 .. Actuators.Number_Of_Actuators) :=
      (others => Actuators.Motor_Min_Angular_Velocity);

    Angular_Velocities : Real_Vector (1 .. Actuators.Number_Of_Actuators) :=
      (others => Actuators.Motor_Min_Angular_Velocity);

  end Agent;

end Actuators.Motors;
```

```

-- actuators-motors.adb

package body Actuators.Motors is

    -----
    -- Agent --
    -----

    protected body Agent is

        -----
        -- Get_Angular_Velocity_References --
        -----

        function Get_Angular_Velocity_References return Real_Vector is
        begin
            return Agent.Angular_Velocity_References;
        end Get_Angular_Velocity_References;

        -----
        -- Get Angular Velocities --
        -----

        function Get_Angular_Velocities return Real_Vector is
        begin
            return Agent.Angular_Velocities;
        end Get_Angular_Velocities;

        -----
        -- Put_Angular_Velocity_References --
        -----

        procedure Put_Angular_Velocity_References (Angular_Velocity_References : Real_Vector) is
        begin

            Agent.Angular_Velocity_References := Utils.Clamp (Angular_Velocity_References,
                                                                Actuators.Motor_Min_Angular_Velocity,
                                                                Actuators.Motor_Max_Angular_Velocity);

        end Put_Angular_Velocity_References;

        -----
        -- Put_Angular_Velocities --
        -----

        procedure Put_Angular_Velocities (Angular_Velocities : Real_Vector) is
        begin

            Agent.Angular_Velocities := Utils.Clamp (Angular_Velocities,
                                                       Actuators.Motor_Min_Angular_Velocity,
                                                       Actuators.Motor_Max_Angular_Velocity);

        end Put_Angular_Velocities;

    end Agent;

end Actuators.Motors;

```

El paquete *Actuators.Motors\_Updater*, especifica e implementa la tarea periódica *Motors\_Updater*:

```
-- actuators-motors_updater.ads

package Actuators.Motors_Updater is

    type Bundle (ESCs : Actuators.Any_ESCs_Reader;
                 Motors : Any_Motors_Operations) is new Cyclics.Bundle with null record;

    procedure On_Start (Self : in out Bundle);

    procedure On_Loop (Self : in out Bundle; Elapsed_Time : Time_Span);

    procedure On_End (Self : in out Bundle);

end Actuators.Motors_Updater;

-- actuators-motors_updater.adb

package body Actuators.Motors_Updater is

    -----
    -- On_Start --
    -----

    procedure On_Start (Self : in out Bundle) is
    begin
        null;
    end On_Start;
```

```

-----
-- On_Loop --
-----

procedure On_Loop (Self : in out Bundle; Elapsed_Time : Time_Span) is
    ESCs_Pulse_Widths : Real_Vector (1 .. Actuators.Number_Of_Actuators);
    Motors_New_Angular_Velocity_References : Real_Vector (1 .. Actuators.Number_Of_Actuators);
    A : constant Float := -Log (0.02) / Actuators.Motors_Settling_Time;
    Sample_Time : constant Float := Float (Self.Release_Interval / Microseconds (1)) * 0.000001;
    Alpha : constant Float := Exp (-A * Sample_Time);
    Beta : constant Float := 1.0 - Alpha;

begin
    -- Map pulse widths to angular velocity references.
    ESCs_Pulse_Widths := Self.ESCs.Get_Pulse_Widths;

    for I in Motors.New_Angular_Velocity_References'First ..
Motors_New_Angular_Velocity_References'Last loop
        Motors_New_Angular_Velocity_References (I) := Utils.Linear_Interpolation (ESCs_Pulse_Widths
(I),
Actuators.ESC_Min_Pulse_Width,
Actuators.ESC_Max_Pulse_Width,
Actuators.Motor_Min_Angular_Velocity,
Actuators.Motor_Max_Angular_Velocity);
    end loop;

    Self.Motors.Put_Angular_Velocity_References (Motors_New_Angular_Velocity_References);

    -- Update angular velocities.
    Self.Motors.Put_Angular_Velocities (Alpha * Self.Motors.Get_Angular_Velocities +
Beta * Self.Motors.Get_Angular_Velocity_References);

end On_Loop;

-----
-- On_End --
-----

procedure On_End (Self : in out Bundle) is
begin
    null;
end On_End;

end Actuators.Motors_Updater;

```

### 5.4.6 Multicóptero

El paquete *Multicopter* define las interfaces *Motion\_Reader* y *Motion\_Operations*. Además, también se especifican en él algunas constantes. Estas constantes se detallan en el apartado de pruebas. El código es:

```
-- multicopter.ads

with Ada.Numerics.Elementary_Functions; use Ada.Numerics.Elementary_Functions;
with Ada.Numerics.Real_Arrays; use Ada.Numerics.Real_Arrays;
with Ada.Real_Time; use Ada.Real_Time;

with System; use System;

with Actuators;
with Cyclics;
with Utils;

package Multicopter is

    -----
    -- Constants --
    -----

    Mass : constant Float := 0.848;

    L : constant Float := 0.225;

    B : constant Float := 0.00000298;

    K : constant Float := 0.000000114;

    Alpha : constant Float := (Sqrt (2.0) / 2.0) * L * B;

    T_Matrix : constant Real_Matrix (1 .. 6, 1 .. 4) := ((0.0, 0.0, 0.0, 0.0),
                                                         (0.0, 0.0, 0.0, 0.0),
                                                         (B, B, B, B),
                                                         (-Alpha, +Alpha, +Alpha, -Alpha),
                                                         (-Alpha, -Alpha, +Alpha, +Alpha),
                                                         (-K, +K, -K, +K));

    Inertia_Tensor: constant Real_Matrix (1 .. 3, 1 .. 3) := ((0.008658454666667, 0.0, 0.0),
                                                             (0.0, 0.009249857333333, 0.0),
                                                             (0.0, 0.0, 0.00077032266));

    MRB_Matrix : constant Real_Matrix (1 .. 6, 1 .. 6) := ((Mass, 0.0, 0.0, 0.0, 0.0, 0.0),
                                                         (0.0, Mass, 0.0, 0.0, 0.0, 0.0),
                                                         (0.0, 0.0, Mass, 0.0, 0.0, 0.0),
                                                         (0.0, 0.0, 0.0, Inertia_Tensor (1, 1), 0.0,
0.0),
                                                         (0.0, 0.0, 0.0, 0.0, Inertia_Tensor (2, 2),
0.0),
                                                         (0.0, 0.0, 0.0, 0.0, 0.0, Inertia_Tensor
(3, 3)));

    MRB_Matrix_Inverse : constant Real_Matrix (1 .. 6, 1 .. 6) := Inverse (MRB_Matrix);

    -----
    -- Motion --
    -----

    -- Motion_Reader interface.

    type Motion_Reader is synchronized interface;

    function Get_Forces_And_Moments (Self : Motion_Reader) return Real_Vector is abstract;

    function Get_Linear_And_Angular_Velocities (Self : Motion_Reader) return Real_Vector is abstract;

    function Get_Position_And_Orientation (Self : Motion_Reader) return Real_Vector is abstract;

    type Any_Motion_Reader is access all Motion_Reader'Class;

    -- Motion_Operations interface.

    type Motion_Operations is synchronized interface and Motion_Reader;
```

```

procedure Put_Forces_And_Moments (Self : in out Motion_Operations; Forces_And_Moments :
Real_Vector) is abstract;

procedure Put_Linear_And_Angular_Velocities (Self : in out Motion_Operations;
Linear_And_Angular_Velocities : Real_Vector) is
abstract;

procedure Put_Position_And_Orientation (Self : in out Motion_Operations;
Position_And_Orientation : Real_Vector) is abstract;

function Get_C_Matrix (Self : Motion_Operations) return Real_Matrix is abstract;
function Get_J_1_Matrix (Self : Motion_Operations) return Real_Matrix is abstract;
function Get_J_2_Matrix (Self : Motion_Operations) return Real_Matrix is abstract;
function Get_J_Matrix (Self : Motion_Operations) return Real_Matrix is abstract;

function Get_Global_To_Local_Vector_Matrix (Self : Motion_Operations) return Real_Matrix is
abstract;

type Any_Motion_Operations is access all Motion_Operations'Class;
end Multicopter;

```

El paquete *Multicopter.Motion* implementa la interfaz *Motion\_Operations* usando un objeto protegido:

```

-- multicopter-motion.ads

package Multicopter.Motion is

  protected type Agent (Ceiling: Priority) with Priority => Ceiling is new Motion_Operations with

    overriding function Get_Forces_And_Moments return Real_Vector;

    overriding function Get_Linear_And_Angular_Velocities return Real_Vector;

    overriding function Get_Position_And_Orientation return Real_Vector;

    overriding procedure Put_Forces_And_Moments (Forces_And_Moments : Real_Vector);

    overriding procedure Put_Linear_And_Angular_Velocities (Linear_And_Angular_Velocities :
Real_Vector);

    overriding procedure Put_Position_And_Orientation (Position_And_Orientation : Real_Vector);

    overriding function Get_C_Matrix return Real_Matrix;

    overriding function Get_J_1_Matrix return Real_Matrix;

    overriding function Get_J_2_Matrix return Real_Matrix;

    overriding function Get_J_Matrix return Real_Matrix;

    overriding function Get_Global_To_Local_Vector_Matrix return Real_Matrix;

  private

    Forces_And_Moments : Real_Vector (1 .. 6) := (others => 0.0);

    Linear_And_Angular_Velocities : Real_Vector (1 .. 6) := (others => 0.0);

    Position_And_Orientation : Real_Vector (1 .. 6) := (others => 0.0);

  end Agent;

end Multicopter.Motion;

```

```

-- multicopter-motion.adb

package body Multicopter.Motion is

    -----
    -- Agent --
    -----

    protected body Agent is

        -----
        -- Get_Forces_And_Moments --
        -----

        function Get_Forces_And_Moments return Real_Vector is
        begin
            return Agent.Forces_And_Moments;
        end Get_Forces_And_Moments;

        -----
        -- Get Linear And Angular Velocities --
        -----

        function Get_Linear_And_Angular_Velocities return Real_Vector is
        begin
            return Agent.Linear_And_Angular_Velocities;
        end Get_Linear_And_Angular_Velocities;

        -----
        -- Get_Position_And_Orientation --
        -----

        function Get_Position_And_Orientation return Real_Vector is
        begin
            return Agent.Position_And_Orientation;
        end Get_Position_And_Orientation;

        -----
        -- Put_Forces_And_Moments --
        -----

        procedure Put_Forces_And_Moments (Forces_And_Moments : Real_Vector) is
        begin
            Agent.Forces_And_Moments := Forces_And_Moments;
        end Put_Forces_And_Moments;

        -----
        -- Put_Linear_And_Angular_Velocities --
        -----

        procedure Put_Linear_And_Angular_Velocities (Linear_And_Angular_Velocities : Real_Vector) is
        begin
            Agent.Linear_And_Angular_Velocities := Linear_And_Angular_Velocities;
        end Put_Linear_And_Angular_Velocities;

        -----
        -- Put_Position_And_Orientation --
        -----

        procedure Put_Position_And_Orientation (Position_And_Orientation : Real_Vector) is
        begin
            Agent.Position_And_Orientation := Position_And_Orientation;
        end Put_Position_And_Orientation;

        -----
        -- Get_C_Matrix --
        -----

        function Get_C_Matrix return Real_Matrix is

            U : constant Float := Agent.Linear_And_Angular_Velocities (1);
            V : constant Float := Agent.Linear_And_Angular_Velocities (2);
            W : constant Float := Agent.Linear_And_Angular_Velocities (3);
            P : constant Float := Agent.Linear_And_Angular_Velocities (4);
            Q : constant Float := Agent.Linear_And_Angular_Velocities (5);
            R : constant Float := Agent.Linear_And_Angular_Velocities (6);

```

```

M : constant Float := Multicopter.Mass;

I_X : constant Float := Multicopter.Inertia_Tensor (1, 1);
I_Y : constant Float := Multicopter.Inertia_Tensor (2, 2);
I_Z : constant Float := Multicopter.Inertia_Tensor (3, 3);

C_Matrix : constant Real_Matrix (1 .. 6, 1 .. 6) := ((0.0, 0.0, 0.0, 0.0, M * W, -M * V),
(0.0, 0.0, 0.0, -M * W, 0.0, M * U),
(0.0, 0.0, 0.0, M * V, -M * U, 0.0),
(0.0, M * W, -M * V, 0.0, I_Z * R, -I_Y
* Q),
(-M * W, 0.0, M * U, -I_Z * R, 0.0, I_X
* P),
(M * V, -M * U, 0.0, I_Y * Q, -I_X * P,
0.0));

begin

    return C_Matrix;

end Get_C_Matrix;

-----
-- Get_J_1_Matrix --
-----

function Get_J_1_Matrix return Real_Matrix is

    Phi : constant Float := Agent.Position_And_Orientation (4);
    Theta : constant Float := Agent.Position_And_Orientation (5);
    Psi : constant Float := Agent.Position_And_Orientation (6);

    S_Phi : constant Float := Sin (Phi);
    C_Phi : constant Float := Cos (Phi);

    S_Theta : constant Float := Sin (Theta);
    C_Theta : constant Float := Cos (Theta);

    S_Psi : constant Float := Sin (Psi);
    C_Psi : constant Float := Cos (Psi);

    J_1_Matrix : constant Real_Matrix (1 .. 3, 1 .. 3) := ((C_Psi * C_Theta,
-S_Psi * C_Phi + C_Psi * S_Theta *
S_Phi,
S_Psi * S_Phi + C_Psi * C_Phi *
S_Theta),
(S_Psi * C_Theta,
C_Psi * C_Phi + S_Phi * S_Theta *
-C_Psi * S_Phi + S_Theta * S_Psi *
(-S_Theta,
C_Theta * S_Phi,
C_Theta * C_Phi));

begin

    return J_1_Matrix;

end Get_J_1_Matrix;

-----
-- Get_J_2_Matrix --
-----

function Get_J_2_Matrix return Real_Matrix is

    Phi : constant Float := Agent.Position_And_Orientation (4);
    Theta : constant Float := Agent.Position_And_Orientation (5);
    Psi : constant Float := Agent.Position_And_Orientation (6);

    Sin_Phi : constant Float := Sin (Phi);
    Cos_Phi : constant Float := Cos (Phi);
    Cos_Theta : constant Float := Cos (Theta);
    Tan_Theta : constant Float := Tan (Theta);

```



```

J_2_Matrix : constant Real_Matrix (1 .. 3, 1 .. 3) := ((1.0, Sin_Phi * Tan_Theta, Cos_Phi *
Tan_Theta),
                                                    (0.0, Cos_Phi, -Sin_Phi),
                                                    (0.0, Sin_Phi / Cos_Theta, Cos_Phi /
Cos_Theta));

begin
    return J_2_Matrix;
end Get_J_2_Matrix;

-----
-- Get_J_Matrix --
-----

function Get_J_Matrix return Real_Matrix is
    J_1_Matrix : constant Real_Matrix := Get_J_1_Matrix;
    J_2_Matrix : constant Real_Matrix := Get_J_2_Matrix;
    J_Matrix : Real_Matrix (1 .. 6, 1 .. 6);

begin
    J_Matrix := (others => (others => 0.0));

    for I in 1 .. 3 loop
        for J in 1 .. 3 loop
            J_Matrix (I, J) := J_1_Matrix (I, J);
        end loop;
    end loop;

    for I in 4 .. 6 loop
        for J in 4 .. 6 loop
            J_Matrix (I, J) := J_2_Matrix (I - 3, J - 3);
        end loop;
    end loop;

    return J_Matrix;
end Get_J_Matrix;

-----
-- Get_Global_To_Local_Vector_Matrix --
-----

function Get_Global_To_Local_Vector_Matrix return Real_Matrix is
    J_1_Matrix_Inverse : constant Real_Matrix (1 .. 3, 1 .. 3) := Transpose
(Agent.Get_J_1_Matrix);
    Global_To_Local_Vector_Matrix : Real_Matrix (1 .. 6, 1 .. 6) := (others => (others => 0.0));

begin
    for I in 1 .. 3 loop
        for J in 1 .. 3 loop
            Global_To_Local_Vector_Matrix (I, J) := J_1_Matrix_Inverse (I, J);
        end loop;
    end loop;

    return Global_To_Local_Vector_Matrix;
end

```

```
        end Get_Global_To_Local_Vector_Matrix;  
    end Agent;  
end Multicopter.Motion;
```

El paquete *Multicopter.Motion\_Updater*, especifica e implementa la tarea periódica *Motion\_Updater*:

```
-- multicopter-motion_updater.ads

package Multicopter.Motion_Updater is

  type Bundle (Motors : Actuators.Any_Motors_Reader; Motion : Any_Motion_Operations) is new
    Cyclics.Bundle with null record;

  procedure On_Start (Self : in out Bundle);

  procedure On_Loop (Self : in out Bundle; Elapsed_Time : Time_Span);

  procedure On_End (Self : in out Bundle);

end Multicopter.Motion_Updater;

-- multicopter-motion_updater.adb

package body Multicopter.Motion_Updater is

  -----
  -- On_Start --
  -----

  procedure On_Start (Self : in out Bundle) is
  begin
    null;
  end On_Start;

  -----
  -- On_Loop --
  -----

  procedure On_Loop (Self : in out Bundle; Elapsed_Time : Time_Span) is

    Sample_Time : constant Float := Float (Self.Release_Interval / Milliseconds (1)) * 0.001;

    New_Forces_And_Moments : Real_Vector (1 .. 6);

    New_Linear_And_Angular_Velocities : Real_Vector (1 .. 6);

    New_Position_And_Orientation : Real_Vector (1 .. 6);

    Angular_Velocities : Real_Vector (1 .. Actuators.Number_Of_Actuators);

    Angular_Velocities_Squared : Real_Vector (1 .. Actuators.Number_Of_Actuators);

    Gravitational_Acceleration : constant Float := 9.80665;

    Global_Gravity_Vector : constant Real_Vector (1 .. 6) := (3 => - Multicopter.Mass *
      Gravitational_Acceleration,
                                                                others => 0.0);

  begin

    -- Motors.

    Angular_Velocities := Self.Motors.Get_Angular_Velocities;

    for I in Angular_Velocities_Squared'Range loop
      Angular_Velocities_Squared (I) := Angular_Velocities (I) ** 2.0;
    end loop;

    -- Forces and moments.

    New_Forces_And_Moments := Multicopter.T_Matrix * Angular_Velocities_Squared +
      Self.Motion.Get_Global_To_Local_Vector_Matrix * Global_Gravity_Vector;

    -- Linear and angular velocities.
```

```

declare

    Term_1 : Real_Vector (1 .. 6);

    Term_2 : Real_Vector (1 .. 6);

begin

    Term_1 := Self.Motion.Get_Forces_And_Moments * Sample_Time;

    Term_2 := Self.Motion.Get_C_Matrix * Self.Motion.Get_Linear_And_Angular_Velocities *
Sample_Time;

    New_Linear_And_Angular_Velocities := Self.Motion.Get_Linear_And_Angular_Velocities +
        MRB_Matrix_Inverse * (Term_1 - Term_2);

end;

-- Position and orientation.

New_Position_And_Orientation := Self.Motion.Get_Position_And_Orientation +
    (Self.Motion.Get_J_Matrix * Self.Motion.Get_Linear_And_Angular_Velocities) * Sample_Time;

-- Update.

Self.Motion.Put_Forces_And_Moments (New_Forces_And_Moments);
Self.Motion.Put_Linear_And_Angular_Velocities (New_Linear_And_Angular_Velocities);
Self.Motion.Put_Position_And_Orientation (New_Position_And_Orientation);

end On_Loop;

-----
-- On_End --
-----

procedure On_End (Self : in out Bundle) is
begin
    null;
end On_End;

end Multicopter.Motion_Updater;

```

### 5.4.7 Sensores

El paquete *Sensors* define las interfaces *Absolute\_Orientation\_Sensor\_Reader* y *Absolute\_Orientation\_Sensor\_Operations*:

```
-- sensors.ads

with Ada.Real_Time; use Ada.Real_Time;

with System; use System;

with Cyclics;
with Multicopter;

package Sensors is

    -----
    -- Absolute orientation sensor --
    -----

    -- Absolute_Orientation_Sensor_Reader interface.

    type Absolute_Orientation_Sensor_Reader is synchronized interface;

    function Get_Roll (Self : Absolute_Orientation_Sensor_Reader) return Float is abstract;

    function Get_Pitch (Self : Absolute_Orientation_Sensor_Reader) return Float is abstract;

    function Get_Yaw_Rate (Self : Absolute_Orientation_Sensor_Reader) return Float is abstract;

    type Any_Absolute_Orientation_Sensor_Reader is access all
    Absolute_Orientation_Sensor_Reader'Class;

    -- Absolute_Orientation_Sensor_Operations interface.

    type Absolute_Orientation_Sensor_Operations is synchronized interface and
    Absolute_Orientation_Sensor_Reader;

    procedure Put_Roll (Self : in out Absolute_Orientation_Sensor_Operations; Roll : Float) is
    abstract;

    procedure Put_Pitch (Self : in out Absolute_Orientation_Sensor_Operations; Pitch : Float) is
    abstract;

    procedure Put_Yaw_Rate (Self : in out Absolute_Orientation_Sensor_Operations; Yaw_Rate : Float) is
    abstract;

    type Any_Absolute_Orientation_Sensor_Operations is access all
    Absolute_Orientation_Sensor_Operations'Class;

end Sensors;
```

El paquete *Sensors.Absolute\_Orientation\_Sensor* implementa la interfaz *Absolute\_Orientation\_Sensor\_Operations* usando un objeto protegido:

```
-- sensors-absolute_orientation_sensor.ads

package Sensors.Absolute_Orientation_Sensor is

  protected type Agent (Ceiling: Priority) with Priority => Ceiling is new
    Absolute_Orientation_Sensor_Operations with

    overriding function Get_Roll return Float;

    overriding function Get_Pitch return Float;

    overriding function Get_Yaw_Rate return Float;

    overriding procedure Put_Roll (Roll : Float);

    overriding procedure Put_Pitch (Pitch : Float);

    overriding procedure Put_Yaw_Rate (Yaw_Rate : Float);

  private

    Roll : Float := 0.0;

    Pitch : Float := 0.0;

    Yaw_Rate : Float := 0.0;

  end Agent;

end Sensors.Absolute_Orientation_Sensor;
```

```

-- sensors-absolute_orientation_sensor.adb

package body Sensors.Absolute_Orientation_Sensor is

    -----
    -- Agent --
    -----

    protected body Agent is

        -----
        -- Get_Roll --
        -----

        function Get_Roll return Float is
        begin
            return Agent.Roll;
        end Get_Roll;

        -----
        -- Get Pitch --
        -----

        function Get_Pitch return Float is
        begin
            return Agent.Pitch;
        end Get_Pitch;

        -----
        -- Get_Yaw_Rate --
        -----

        function Get_Yaw_Rate return Float is
        begin
            return Agent.Yaw_Rate;
        end Get_Yaw_Rate;

        -----
        -- Put_Roll --
        -----

        procedure Put_Roll (Roll : Float) is
        begin
            Agent.Roll := Roll;
        end Put_Roll;

        -----
        -- Put_Pitch --
        -----

        procedure Put_Pitch (Pitch : Float) is
        begin
            Agent.Pitch := Pitch;
        end Put_Pitch;

        -----
        -- Put_Yaw_Rate --
        -----

        procedure Put_Yaw_Rate (Yaw_Rate : Float) is
        begin
            Agent.Yaw_Rate := Yaw_Rate;
        end Put_Yaw_Rate;

    end Agent;

end Sensors.Absolute_Orientation_Sensor;

```

El paquete *Sensors.Absolute\_Orientation\_Sensor\_Updater*, especifica e implementa la tarea periódica *Absolute\_Orientation\_Sensor\_Updater*:

```
-- sensors-absolute_orientation_sensor_updater.ads

package Sensors.Absolute_Orientation_Sensor_Updater is

    type Bundle (Motion : Multicopter.Any_Motion_Reader;
                  Absolute_Orientation_Sensor : Any_Absolute_Orientation_Sensor_Operations) is new
    Cyclics.Bundle with null record;

    procedure On_Start (Self : in out Bundle);

    procedure On_Loop (Self : in out Bundle; Elapsed_Time : Time_Span);

    procedure On_End (Self : in out Bundle);

end Sensors.Absolute_Orientation_Sensor_Updater;

-- sensors-absolute_orientation_sensor_updater.adb

package body Sensors.Absolute_Orientation_Sensor_Updater is

    -----
    -- On_Start --
    -----

    procedure On_Start (Self : in out Bundle) is
    begin
        null;
    end On_Start;

    -----
    -- On_Loop --
    -----

    procedure On_Loop (Self : in out Bundle; Elapsed_Time : Time_Span) is
    begin

        Self.Absolute_Orientation_Sensor.Put_Roll (Self.Motion.Get_Position_And_Orientation (4));
        Self.Absolute_Orientation_Sensor.Put_Pitch (Self.Motion.Get_Position_And_Orientation (5));
        Self.Absolute_Orientation_Sensor.Put_Yaw_Rate (Self.Motion.Get_Linear_And_Angular_Velocities
(6));

        end On_Loop;

    -----
    -- On_End --
    -----

    procedure On_End (Self : in out Bundle) is
    begin
        null;
    end On_End;

end Sensors.Absolute_Orientation_Sensor_Updater;
```



### 5.4.8 Logging

El paquete *Logging* no define ninguna interfaz, por lo que simplemente importamos aquellos paquetes de los que dependa. La especificación de este paquete es:

```
-- logging.ads

with Ada.Numerics.Real_Arrays; use Ada.Numerics.Real_Arrays;
with Ada.Real_Time; use Ada.Real_Time;
with Ada.Text_IO; use Ada.Text_IO;

with Actuators;
with Cyclics;
with Flight_Controller;
with Multicopter;
with Sensors;

package Logging is

end Logging;

-- logging-logger.ads

package Logging.Logger is

  type Bundle (File_Handler : access File_Type;
               File_Path : access String;
               References : Flight_Controller.Any_References_Reader;
               ESCs : Actuators.Any_ESCs_Reader;
               Motors : Actuators.Any_Motors_Reader;
               Motion : Multicopter.Any_Motion_Reader;
               Absolute_Orientation_Sensor : Sensors.Any_Absolute_Orientation_Sensor_Reader)

  is new Cyclics.Bundle with null record;

  procedure On_Start (Self : in out Bundle);

  procedure On_Loop (Self : in out Bundle; Elapsed_Time : Time_Span);

  procedure On_End (Self : in out Bundle);

end Logging.Logger;
```

El paquete *Logging.Logger*, especifica e implementa la tarea periódica *Logger*:

```
-- logging-logger.adb

package body Logging.Logger is

    -----
    -- On_Start --
    -----

    procedure On_Start (Self : in out Bundle) is
    begin

        Create (Self.File_Handler.all, Out_File, Self.File_Path.all);

        Put (Self.File_Handler.all, "Time,");

        Put (Self.File_Handler.all,
            "Throttle_Reference,Roll_Reference,Pitch_Reference,Yaw_Rate_Reference,");

        Put (Self.File_Handler.all, "ESC_1,ESC_2,ESC_3,ESC_4,");

        Put (Self.File_Handler.all, "Motor_1,Motor_2,Motor_3,Motor_4,");

        Put (Self.File_Handler.all, "X,Y,Z,K,M,N,u,v,w,p,q,r,x,y,z,phi,theta,psi,");

        Put (Self.File_Handler.all, "Sensor_Roll,Sensor_Pitch,Sensor_Yaw_Rate");

        New_Line (Self.File_Handler.all);

    end On_Start;

    -----
    -- On_Loop --
    -----

    procedure On_Loop (Self : in out Bundle; Elapsed_Time : Time_Span) is

        Elapsed_Time_Float : constant Float := Float (Elapsed_Time / Milliseconds (1)) * 0.001;

        Throttle_Reference : constant Float := Self.References.Get_Throttle;
        Roll_Reference      : constant Float := Self.References.Get_Roll;
        Pitch_Reference      : constant Float := Self.References.Get_Pitch;
        Yaw_Rate_Reference   : constant Float := Self.References.Get_Yaw_Rate;

        ESCs_Pulse_Widths : constant Real_Vector := Self.ESCs.Get_Pulse_Widths;

        Motors_Angular_Velocities : constant Real_Vector := Self.Motors.Get_Angular_Velocities;

        Multicopter_Forces_And_Moments : constant Real_Vector := Self.Motion.Get_Forces_And_Moments;
        Multicopter_Linear_And_Angular_Velocities : constant Real_Vector :=
            Self.Motion.Get_Linear_And_Angular_Velocities;
        Multicopter_Position_And_Orientation : constant Real_Vector :=
            Self.Motion.Get_Position_And_Orientation;

        Sensor_Roll : constant Float := Self.Absolute_Orientation_Sensor.Get_Roll;
        Sensor_Pitch : constant Float := Self.Absolute_Orientation_Sensor.Get_Pitch;
        Sensor_Yaw_Rate : constant Float := Self.Absolute_Orientation_Sensor.Get_Yaw_Rate;

    begin

        -- Time.

        Put (Self.File_Handler.all, Elapsed_Time_Float'Image); Put (Self.File_Handler.all, ",");

        -- References.

        Put (Self.File_Handler.all, Throttle_Reference'Image); Put (Self.File_Handler.all, ",");
        Put (Self.File_Handler.all, Roll_Reference'Image); Put (Self.File_Handler.all, ",");
        Put (Self.File_Handler.all, Pitch_Reference'Image); Put (Self.File_Handler.all, ",");
        Put (Self.File_Handler.all, Yaw_Rate_Reference'Image); Put (Self.File_Handler.all, ",");

        -- ESCs.

        for I in ESCs_Pulse_Widths'Range loop
```

```

        Put (Self.File_Handler.all, ESCs_Pulse_Widths (I)'Image); Put (Self.File_Handler.all, ",");

    end loop;

    -- Motors.

    for I in Motors_Angular_Velocities'Range loop

        Put (Self.File_Handler.all, Motors_Angular_Velocities (I)'Image); Put
(Self.File_Handler.all, ",");

    end loop;

    -- Motion.

    for I in Multicopter_Forces_And_Moments'Range loop

        Put (Self.File_Handler.all, Multicopter_Forces_And_Moments (I)'Image); Put
(Self.File_Handler.all, ",");

    end loop;

    for I in Multicopter_Linear_And_Angular_Velocities'Range loop

        Put (Self.File_Handler.all, Multicopter_Linear_And_Angular_Velocities (I)'Image); Put
(Self.File_Handler.all, ",");

    end loop;

    for I in Multicopter_Position_And_Orientation'Range loop

        Put (Self.File_Handler.all, Multicopter_Position_And_Orientation (I)'Image); Put
(Self.File_Handler.all, ",");

    end loop;

    -- Sensor.

    Put (Self.File_Handler.all, Sensor_Roll'Image); Put (Self.File_Handler.all, ",");
    Put (Self.File_Handler.all, Sensor_Pitch'Image); Put (Self.File_Handler.all, ",");
    Put (Self.File_Handler.all, Sensor_Yaw_Rate'Image);

    -- New line.

    New_Line (Self.File_Handler.all);

end On_Loop;

-----
-- On_End --
-----

procedure On_End (Self : in out Bundle) is
begin
    Close (Self.File_Handler.all);
end On_End;

end Logging_Logger;

```

#### 5.4.9 Configuración

Para ejecutar el sistema, necesitamos una implementación de la tarea *Maneuvering.Driver* (vista anteriormente), un archivo de configuración, y un archivo que contenga el método principal. En el fichero de configuración se crean todas las variables necesarias; parámetros de la simulación, configuración de los *PIDs*, periodos de las tareas, prioridades de las tareas, techos de prioridad de los objetos protegidos, los objetos protegidos, los controladores *PID* y las tareas:

```
-- configuration.ads

with Ada.Real_Time; use Ada.Real_Time;
with Ada.Text_IO; use Ada.Text_IO;

with System; use System;

with Actuators.Electronic_Speed_Controllers;
with Actuators.Motors;
with Actuators.Motors_Updater;
with Cyclics;
with Flight_Controller.Control_Algorithm;
with Flight_Controller.References;
with Logging.Logger;
with Mixers;
with Multicopter.Motion;
with Multicopter.Motion_Updater;
with PID_Controllers;
with Sensors.Absolute_Orientation_Sensor;
with Sensors.Absolute_Orientation_Sensor_Updater;

with Driver;

package Configuration is

    -----
    -- Simulation --
    -----

    Simulation_Time : constant Time_Span := Seconds (10);

    File_Path : aliased String := "./test_1.csv";

    Roll_PID_KP : constant Float := 45.0;
    Roll_PID_KI : constant Float := 1.75;
    Roll_PID_KD : constant Float := 20.0;

    Pitch_PID_KP : constant Float := 45.0;
    Pitch_PID_KI : constant Float := 1.75;
    Pitch_PID_KD : constant Float := 20.0;

    Yaw_Rate_PID_KP : constant Float := 20.0;
    Yaw_Rate_PID_KI : constant Float := 0.05;
    Yaw_Rate_PID_KD : constant Float := 2.0;

    PID_Min_Output : constant Float := -500.0;
    PID_Max_Output : constant Float := +500.0;

    -----
    -- Release Intervals --
    -----

    Motors_Updater_Release_Interval : constant Time_Span := Microseconds (100);

    Motion_Updater_Release_Interval : constant Time_Span := Milliseconds (5);

    Control_Algorithm_Release_Interval : constant Time_Span := Milliseconds (10);

    Absolute_Orientation_Sensor_Updater_Release_Interval : constant Time_Span := Milliseconds (10);

    Logger_Release_Interval : constant Time_Span := Milliseconds (50);

    Driver_Release_Interval : constant Time_Span := Milliseconds (60);

    -----
    -- Priorities --
    -----
```

```

-----

Motors_Updater_Priority : constant Priority := 10;

Motion_Updater_Priority : constant Priority := 9;

Control_Algorithm_Priority : constant Priority := 8;

Absolute_Orientation_Sensor_Updater_Priority : constant Priority := 7;

Logger_Priority : constant Priority := 6;

Driver_Priority : constant Priority := 5;

-----
-- Ceilings --
-----

ESCs_Ceiling : constant Priority := 10;           -- max{8, 10, 5} = 10.
Motors_Ceiling : constant Priority := 10;         -- max{10, 9, 6} = 10.
Motion_Ceiling : constant Priority := 9;          -- max{9, 7, 6} = 9.
References_Ceiling : constant Priority := 8;      -- max{5, 8, 6} = 8.
Absolute_Orientation_Sensor_Ceiling : constant Priority := 8; -- max{7, 8, 6} = 8.

-----
-- Agents --
-----

ESCs_Agent : aliased Actuators.Electronic_Speed_Controllers.Agent (Ceiling => ESCs_Ceiling);
Motors_Agent : aliased Actuators.Motors.Agent (Ceiling => Motors_Ceiling);
Motion_Agent : aliased Multicopter.Motion.Agent (Ceiling => Motion_Ceiling);
References_Agent : aliased Flight_Controller.References.Agent (Ceiling => References_Ceiling);
Absolute_Orientation_Sensor_Agent : aliased Sensors.Absolute_Orientation_Sensor.Agent (Ceiling =>
Absolute_Orientation_Sensor_Ceiling);

-----
-- Other --
-----

PID_Controllers_Sample_Time : constant Float := Float (Control_Algorithm_Release_Interval /
Milliseconds (1)) * 0.001;

Roll_PID : aliased PID_Controllers.Parallel_PID_Controller := PID_Controllers.Create (Roll_PID_KP,
                                                                                       Roll_PID_KI,
                                                                                       Roll_PID_KD,

PID_Controllers_Sample_Time,
PID_Min_Output,
PID_Max_Output);

Pitch_PID : aliased PID_Controllers.Parallel_PID_Controller := PID_Controllers.Create
(Pitch_PID_KP,
Pitch_PID_KI,
Pitch_PID_KD,
PID_Controllers_Sample_Time,
PID_Min_Output,
PID_Max_Output);

Yaw_Rate_PID : aliased PID_Controllers.Parallel_PID_Controller := PID_Controllers.Create
(Yaw_Rate_PID_KP,
Yaw_Rate_PID_KI,

```

```

Yaw_Rate_PID_KD,

PID_Controllers_Sample_Time,

PID_Min_Output,

PID_Max_Output);

    Quadcopter_Cross_Mixer : aliased Mixers.Quadcopter_Cross_Mixer := Mixers.Create
(Actuators.ESC_Min_Pulse_Width,

Actuators.ESC_Max_Pulse_Width);

    File_Handler : aliased File_Type;

-----
-- Bundles --
-----

Driver_Bundle : aliased Driver.Bundle :=

(Task_Priority => Driver_Priority,
Release_Interval => Driver_Release_Interval,
Task_Duration => Simulation_Time,
References => References_Agent'Access);

Control_Algorithm_Bundle : aliased Flight_Controller.Control_Algorithm.Bundle :=

(Task_Priority => Control_Algorithm_Priority,
Release_Interval => Control_Algorithm_Release_Interval,
Task_Duration => Simulation_Time,
Roll_PID => Roll_PID'Access,
Pitch_PID => Pitch_PID'Access,
Yaw_Rate_PID => Yaw_Rate_PID'Access,
Quadcopter_Cross_Mixer => Quadcopter_Cross_Mixer'Access,
References => References_Agent'Access,
ESCs => ESCs_Agent'Access,
Absolute_Orientation_Sensor => Absolute_Orientation_Sensor_Agent'Access);

Motors_Updater_Bundle : aliased Actuators.Motors_Updater.Bundle :=

(Task_Priority => Motors_Updater_Priority,
Release_Interval => Motors_Updater_Release_Interval,
Task_Duration => Simulation_Time,
ESCs => ESCs_Agent'Access,
Motors => Motors_Agent'Access);

Motion_Updater_Bundle : aliased Multicopter.Motion_Updater.Bundle :=

(Task_Priority => Motion_Updater_Priority,
Release_Interval => Motion_Updater_Release_Interval,
Task_Duration => Simulation_Time,
Motors => Motors_Agent'Access,
Motion => Motion_Agent'Access);

Absolute_Orientation_Sensor_Updater_Bundle : aliased
Sensors.Absolute_Orientation_Sensor_Updater.Bundle :=

(Task_Priority => Absolute_Orientation_Sensor_Updater_Priority,
Release_Interval => Absolute_Orientation_Sensor_Updater_Release_Interval,
Task_Duration => Simulation_Time,
Motion => Motion_Agent'Access,
Absolute_Orientation_Sensor => Absolute_Orientation_Sensor_Agent'Access);

Logger_Bundle : aliased Logging.Logger.Bundle :=

(Task_Priority => Logger_Priority,
Release_Interval => Logger_Release_Interval,
Task_Duration => Simulation_Time,
File_Handler => File_Handler'Access,
File_Path => File_Path'Access,
References => References_Agent'Access,
ESCs => ESCs_Agent'Access,
Motors => Motors_Agent'Access,
Motion => Motion_Agent'Access,
Absolute_Orientation_Sensor => Absolute_Orientation_Sensor_Agent'Access);

```

```

-----
-- Tasks --
-----

Driver_Task : Cyclics.Cyclic (Driver_Bundle'Access);

Control_Algorithm_Task : Cyclics.Cyclic (Control_Algorithm_Bundle'Access);

Motors_Updater_Task : Cyclics.Cyclic (Motors_Updater_Bundle'Access);

Motion_Updater_Task : Cyclics.Cyclic (Motion_Updater_Bundle'Access);

Absolute_Orientation_Sensor_Updater_Task : Cyclics.Cyclic
(Absolute_Orientation_Sensor_Updater_Bundle'Access);

Logger_Task : Cyclics.Cyclic (Logger_Bundle'Access);

end Configuration;

```

En el fichero que contiene el método principal, basta con especificar que se usa el perfil de *Ravenscar* e importar el archivo de configuración. El método *Main* no tiene cuerpo:

```

-- main.adb

pragma Profile (Ravenscar);

with Configuration;

procedure Main is
begin
    null;
end Main;

```

## 5.5 Pruebas

### 5.5.1 Parámetros de configuración comunes a todas las pruebas

En la tabla siguiente se pueden ver los parámetros de configuración comunes usados en todas las pruebas.

Parámetro	Valor	Unidades
<b>Controlador de vuelo</b>		
<i>Throttle</i> mínimo	1000	$\mu s$
<i>Throttle</i> máximo	2000	$\mu s$
<i>Roll</i> mínimo	-25	$^{\circ}$
<i>Roll</i> máximo	+25	$^{\circ}$
<i>Pitch</i> mínimo	-25	$^{\circ}$
<i>Pitch</i> máximo	+25	$^{\circ}$
<i>Yaw rate</i> mínimo	-180	$rad/s$
<i>Yaw rate</i> máximo	+180	$rad/s$
<b>Actuadores</b>		
Ancho de pulso mínimo	1000	$\mu s$
Ancho de pulso máximo	2000	$\mu s$
Velocidad angular mínima ( $\gamma_m$ )	0	$rad/s$
Velocidad angular máxima ( $\gamma_M$ )	1240	$rad/s$
Tiempo de estabilización	0.250	$s$
<b>Multicóptero</b>		
Aceleración gravitacional ( $g$ )	9.80665	$m/s^2$
Masa	0.848	$kg$
Longitud de los brazos	0.225	$m$
Constante $b$	0.00000298	-
Constante $k$	0.000000114	-
Momento de inercia en el eje $x$ ( $I_x$ )	0.008658454666667	$kg \cdot m^2$
Momento de inercia en el eje $y$ ( $I_y$ )	0.009249857333333	$kg \cdot m^2$
Momento de inercia en el eje $z$ ( $I_z$ )	0.00077032266	$kg \cdot m^2$
<b>Archivo de configuración</b>		
Tiempo de simulación	10	$s$
<i>Roll</i> PID $K_P$	45	-
<i>Roll</i> PID $K_I$	1.75	-
<i>Roll</i> PID $K_D$	20	-
<i>Roll</i> PID salida mínima	-500	-
<i>Roll</i> PID salida máxima	+500	-
<i>Pitch</i> PID $K_P$	45	-
<i>Pitch</i> PID $K_I$	1.75	-
<i>Pitch</i> PID $K_D$	20	-
<i>Pitch</i> PID salida mínima	-500	-
<i>Pitch</i> PID salida máxima	+500	-
<i>Yaw rate</i> PID $K_P$	20	-
<i>Yaw rate</i> PID $K_I$	0.05	-
<i>Yaw rate</i> PID $K_D$	2	-
<i>Yaw rate</i> PID salida mínima	-500	-
<i>Yaw rate</i> PID salida máxima	+500	-

Tabla 4 - Parámetros comunes a todas las pruebas



A continuación, comentamos brevemente la elección de algunos de estos parámetros:

- Los parámetros del controlador de vuelo se han configurado dando unos valores razonables a las referencias de vuelo mínimas y máximas.
- El parámetro de velocidad angular máxima en los actuadores ( $\omega_M$ ) se ha obtenido suponiendo que se usa una batería de  $14.8\text{ V}$  con unos motores de parámetro  $KV = 800\text{ rpm/V}$ . Por tanto, una estimación inicial para la velocidad angular máxima sería de  $14.8 \cdot 800 = 11840\text{ rpm} \approx 1240\text{ rad/s}$ . Se puede encontrar más información en [10]. Las velocidades angulares máximas de los motores real serían inferiores a la calculada, pues en la estimación usada no se tiene en cuenta el efecto de las hélices.
- Sobre la configuración del multicoptero:
  - La masa y los momentos de inercia del multicoptero se detallan en el anexo 9.3.
  - Se han usado los mismos valores para  $k$  y  $b$  que los usados en [4]. Esto se debe a que no se dispone de los materiales necesarios para estudiar y/o estimar estos parámetros.
- La calibración de los controladores  $PID$  se ha hecho siguiendo heurísticas clásicas.

### 5.5.2 Parámetros de tiempo real

Empezamos asignando prioridades a las tareas usando el método *rate monotonic priority assignment*. De esta manera a menor periodo (mayor frecuencia), mayor prioridad. Los periodos se asignan de manera experimental, dando mayor prioridad a las tareas de simulación:

Tarea	Prioridad	Periodo (ms)
Motors_Updater	10	0.1
Motion_Updater	9	5
Control_Algorithm	8	10
Absolute_Orientation_Sensor_Updater	7	10
Logger	6	50
Driver	5	60

Tabla 5 - Asignación de prioridades

Cabe destacar que las tareas *Control\_Algorithm* y *Absolute\_Orientation\_Sensor\_Updater* tienen el mismo periodo, ya que no tiene sentido ejecutar el bucle de control a mayor frecuencia que la de la actualización del sensor.

Para asignar las prioridades de techo a los objetos protegidos, es útil disponer las tareas y los objetos protegidos de manera que se refleje qué tarea accede a qué objeto protegido:

Tarea \ Objeto protegido	ESCs	Motors	Motion	References	AO Sensor
Motors_Updater	X	X			
Motion_Updater		.	X		
Control_Algorithm	.			X	X
Absolute_Orientation_Sensor_Updater			.		.
Logger	.	.	.	.	.
Driver				.	

Tabla 6 - Uso de objetos protegidos por tareas

La prioridad de techo de un objeto protegido se calcula como la prioridad más alta de todas las tareas que acceden a él. Por ejemplo, en el caso del objeto protegido *Motors*, tenemos que las tareas que acceden a él son *Motors\_Updater* (con prioridad 10), *Motion\_Updater* (con prioridad 9) y *Logger* (con prioridad 5). Por tanto, su prioridad de techo será  $\max\{10,9,5\} = 10$ .

Repitiendo este cálculo para cada objeto protegido, podemos resumir las prioridades de techo en la siguiente tabla:

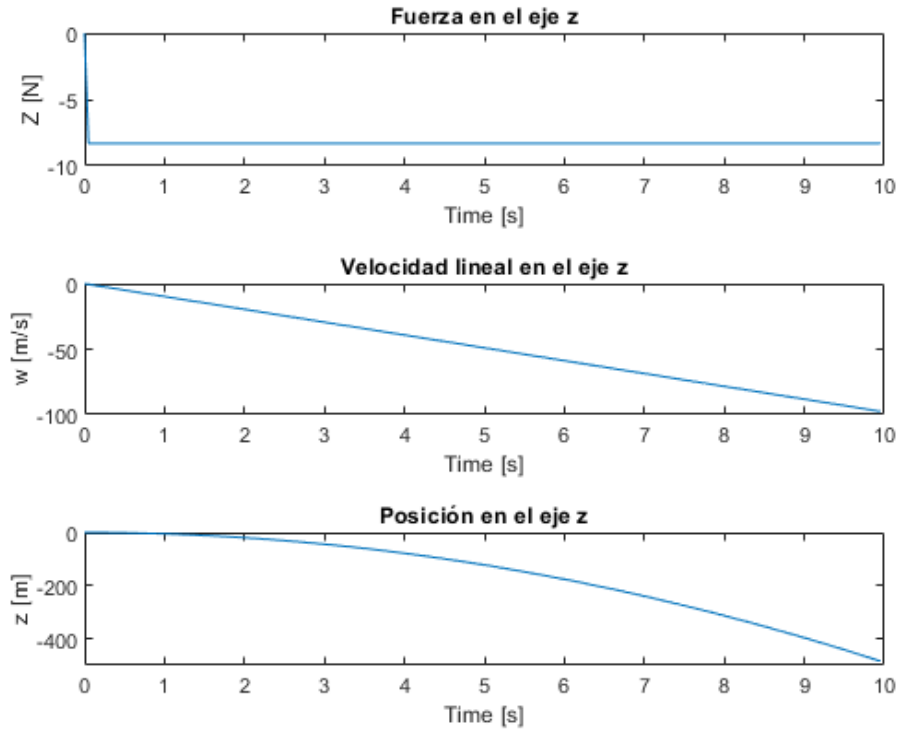
Objeto protegido	<i>Ceiling</i>
<i>ESCs</i>	10
<i>Motors</i>	10
<i>Motion</i>	9
<i>References</i>	8
<i>Absolute_Orientation_Sensor</i>	8

Tabla 7 - Prioridades de techo

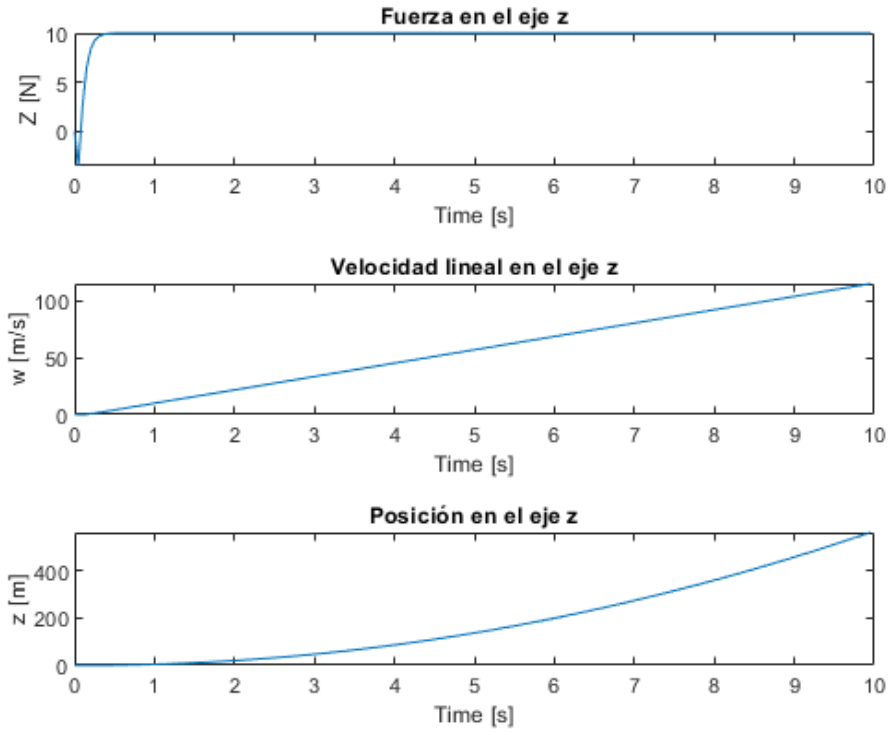
No se ha hecho un análisis de tiempo de respuesta riguroso, ya que este depende en gran medida de la potencia de la máquina en el que se vaya a ejecutar el simulador. No obstante, en el anexo 8.1 se describen los *tests* aplicados para evaluar inicialmente la planificabilidad del simulador.

A continuación, veremos en detalle todas las pruebas que se han hecho al simulador. El código particular de cada prueba se puede encontrar en el anexo 9.4.

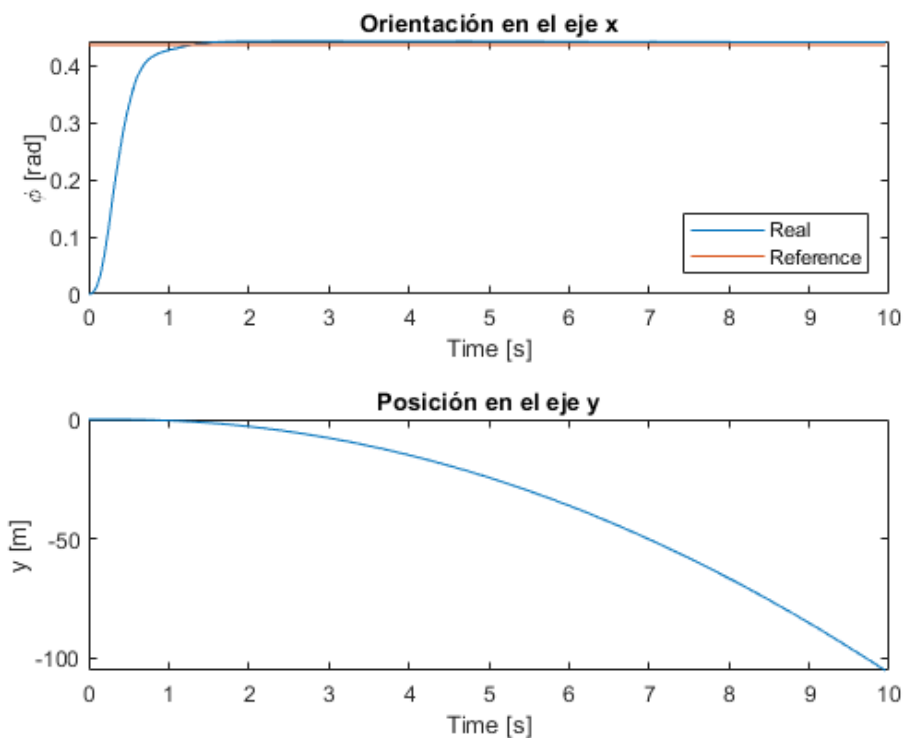
## 5.5.3 Prueba 1: Caída libre

<b>Descripción</b>	Se mantienen los motores parados aplicando la referencia de <i>throttle</i> mínima, <b>1000 <math>\mu</math>s</b> .
<b>Referencias de vuelo</b>	$Throttle = 1000, \phi_D = 0^\circ, \theta_D = 0^\circ$ y $r_D = 0^\circ/s$ .
<b>Resultados esperados</b>	Al mantener los motores parados el vehículo entrará en caída libre, trasladándose en la dirección negativa del eje $z$ .
<b>Parámetros de movimiento de interés</b>	Fuerza en el eje $z$ : $N$ , velocidad lineal en el eje $z$ : $w$ y posición en el eje $z$ : $z$ .
<b>Resultados</b>	 <p style="text-align: center;">Figura 39 – Resultados de la prueba 1</p>
<b>Comentarios</b>	<p>La fuerza obtenida en la simulación es de <math>-8.31604\text{ N}</math> siendo la fuerza esperada <math>F = m \cdot a = 0.848 \cdot (-9.80665) = -8.31604\text{ N}</math>.</p> <p>La velocidad lineal final alcanzada en la simulación es de <math>-97.5285\text{ m/s}</math> siendo la velocidad esperada <math>w(t = 10) = -9.80665 \cdot 10 = -98.665\text{ m/s}</math>.</p> <p>La posición alcanzada al final de la simulación es de <math>-484.716\text{ m}</math> siendo la posición esperada <math>z(t = 10) = 0.5 \cdot (-9.80665) \cdot 10^2 = -490.3325\text{ m}</math>.</p>

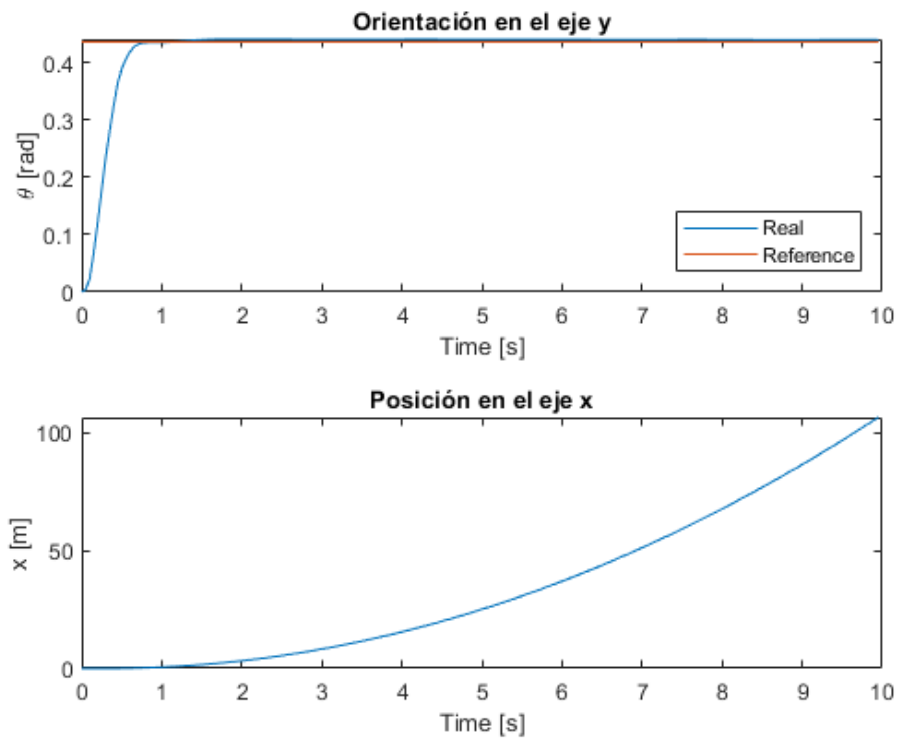
## 5.5.4 Prueba 2: Despegue

<b>Descripción</b>	Se aplica la referencia máxima de <i>throttle</i> , <b>2000 <math>\mu</math>s</b> .
<b>Referencias de vuelo</b>	$Throttle = 2000, \phi_D = 0^\circ, \theta_D = 0^\circ$ y $r_D = 0^\circ/s$ .
<b>Resultados esperados</b>	El vehículo se traslada en la dirección positiva del eje z.
<b>Parámetros de movimiento de interés</b>	Fuerza en el eje z: $N$ , velocidad lineal en el eje z: $w$ y posición en el eje z: $z$ .
<b>Resultados</b>	 <p style="text-align: center;">Figura 40 - Resultados de la prueba 2</p>
<b>Comentarios</b>	<p>La fuerza obtenida en la simulación es de 10.0104 <math>N</math> siendo la fuerza esperada <math>Z = b(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2) + (0.848 \cdot (-9.80665)) = b(4 \cdot 1240^2) + (0.848 \cdot (-9.80665)) = 10.0122</math>.</p> <p>La velocidad lineal final alcanzada en la simulación es de 115.284 <math>m/s</math> siendo la velocidad esperada (suponiendo motores ideales) <math>w(t = 10) = 10.0122/0.848 \cdot 10 = 118.068 \text{ m/s}</math>.</p> <p>La posición alcanzada al final de la simulación es de 562.621 <math>m</math> siendo la posición esperada (suponiendo motores ideales) <math>z(t = 10) = 0.5 \cdot (10.0122/0.848) \cdot 10^2 = 590.342 \text{ m}</math>.</p>

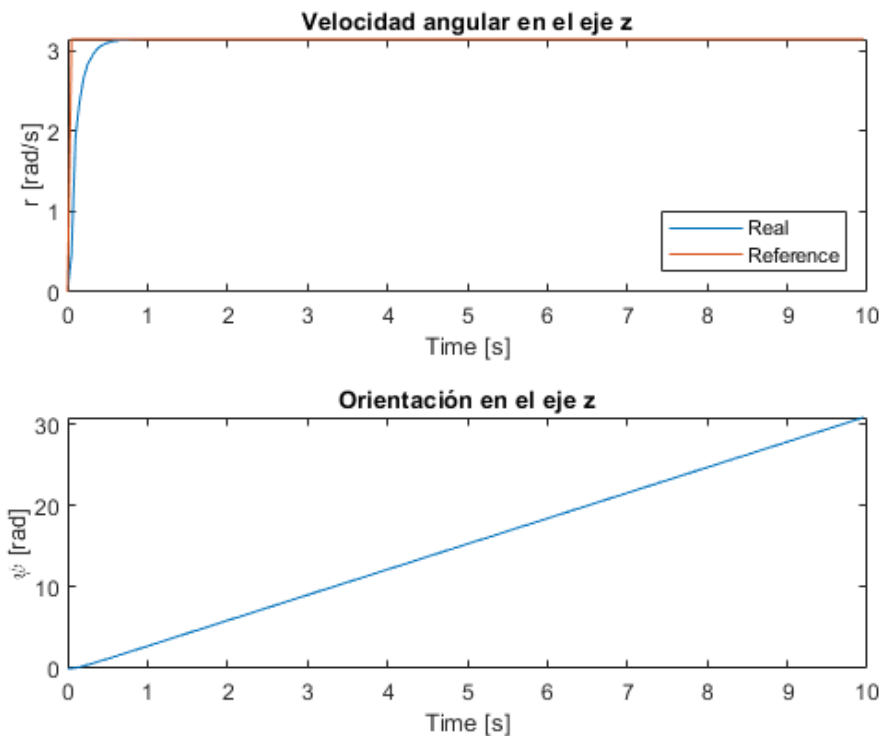
## 5.5.5 Prueba 3: Roll

<b>Descripción</b>	Se establece una referencia de vuelo adecuada para que el vehículo se desplace en la dirección negativa del eje $y$ .
<b>Referencias de vuelo</b>	$Throttle = 1500$ , $\phi_D = 25^\circ = 0.436332 \text{ rad}$ , $\theta_D = 0^\circ$ y $r_D = 0^\circ/s$ .
<b>Resultados esperados</b>	El vehículo conseguir adoptar la referencia de vuelo indicada (orientación de $25^\circ$ en el eje $x$ ), consiguiendo el desplazamiento deseado (a lo largo de la dirección negativa del eje $y$ ).
<b>Parámetros de movimiento de interés</b>	Orientación en el eje $x$ : $\phi$ , referencia de roll: $\phi_D$ y posición en el eje $y$ : $y$ .
<b>Resultados</b>	 <p style="text-align: center;">Figura 41 - Resultados de la prueba 3</p>
<b>Comentarios</b>	<p>Se alcanza el 98 % de la referencia de roll indicada en 1.051 s. En régimen estacionario el roll se establece por encima de la referencia. El último dato de la simulación indica que el error en régimen permanente es de <math>0.005 \text{ rad} = 0.29^\circ</math>.</p> <p>La trayectoria es la esperada.</p>

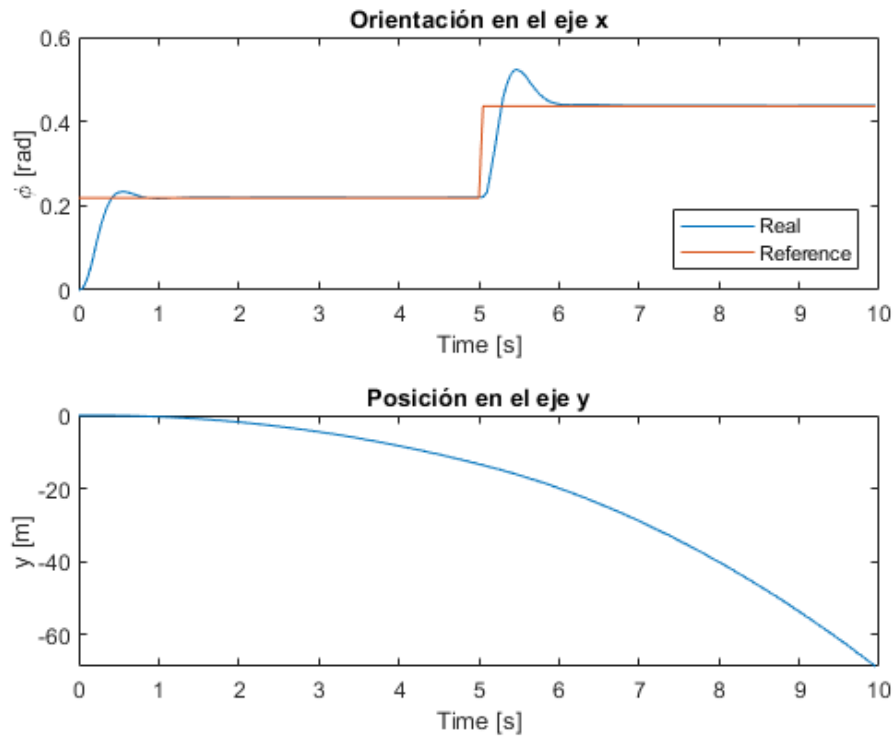
5.5.6 Prueba 4: *Pitch*

<b>Descripción</b>	Se establece una referencia de vuelo adecuada para que el vehículo se desplace en la dirección positiva del eje $x$ .
<b>Referencias de vuelo</b>	$Throttle = 1500$ , $\phi_D = 0^\circ$ , $\theta_D = 25^\circ = 0.436332 \text{ rad}$ y $r_D = 0^\circ/s$ .
<b>Resultados esperados</b>	El vehículo conseguir adoptar la referencia de vuelo indicada (orientación de $25^\circ$ en el eje $y$ ), consiguiendo el desplazamiento deseado (a lo largo de la dirección positiva del eje $x$ ).
<b>Parámetros de movimiento de interés</b>	Orientación en el eje $y$ : $\theta$ , referencia de <i>pitch</i> : $\theta_D$ y posición en el eje $x$ : $x$ .
<b>Resultados</b>	 <p style="text-align: center;">Figura 42 - Resultados de la prueba 4</p>
<b>Comentarios</b>	<p>Se alcanza el 98 % de la referencia de <i>pitch</i> indicada en 0.7 s. En régimen estacionario el <i>pitch</i> se establece por encima de la referencia. El último dato de la simulación indica que el error en régimen permanente es de <math>0.004 \text{ rad} = 0.23^\circ</math>.</p> <p>La trayectoria es la esperada.</p>

5.5.7 Prueba 5: *Yaw rate*

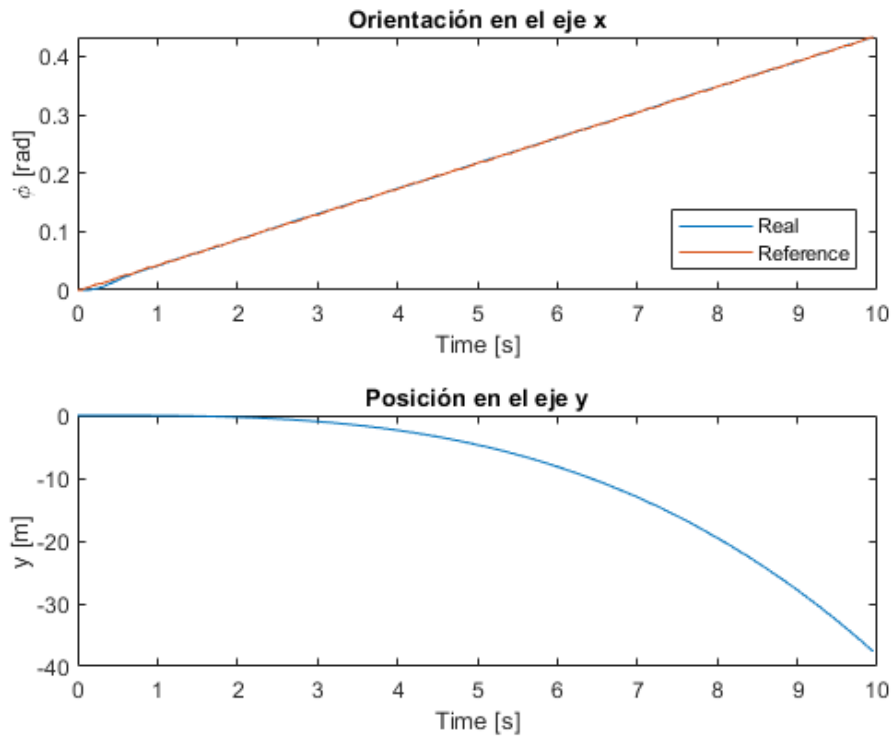
<b>Descripción</b>	Se establece una referencia de vuelo adecuada para que el vehículo rote en la dirección positiva del eje $z$ .
<b>Referencias de vuelo</b>	$Throttle = 1500, \phi_D = 0^\circ, \theta_D = 0$ y $r_D = 180^\circ/s = \pi \text{ rad/s}$ .
<b>Resultados esperados</b>	El vehículo conseguir adoptar la referencia de vuelo indicada (una velocidad de rotación de $180^\circ$ en la dirección positiva del eje $z$ ).
<b>Parámetros de movimiento de interés</b>	Velocidad angular en el eje $z$ : $r$ , referencia de <i>yaw rate</i> : $r_D$ y orientación en el eje $z$ : $\psi$ .
<b>Resultados</b>	 <p style="text-align: center;">Figura 43 - Resultados de la prueba 5</p>
<b>Comentarios</b>	<p>Se alcanza el 98 % de la referencia de <i>yaw rate</i> indicada en 0.5 s. En régimen estacionario el <i>yaw rate</i> se establece por encima de la referencia. El error es prácticamente nulo.</p> <p>Con la velocidad de rotación indicada, y suponiendo motores ideales, se espera que el vehículo de 5 vueltas completas sobre sí mismo, es decir, la orientación en el eje <math>z</math> teórica sería de <math>31.41593 \text{ rad}</math>. El resultado obtenido ha sido de <math>30.884 \text{ rad}</math>.</p>

5.5.8 Prueba 6: Dos referencias de *roll*

<b>Descripción</b>	Se establece una referencia de <i>roll</i> inicial durante los 5 primeros segundos de la simulación. A partir de ese instante de tiempo se establece una referencia de <i>roll</i> del doble de la original.
<b>Referencias de vuelo</b>	En $t = 0$ : $Throttle = 1500$ , $\phi_D = 12.5^\circ = 0.218166 \text{ rad}$ , $\theta_D = 0$ y $r_D = 0^\circ/\text{s}$ . En $t = 5$ : $Throttle = 1500$ , $\phi_D = 25.0^\circ = 0.436332 \text{ rad}$ , $\theta_D = 0$ y $r_D = 0^\circ/\text{s}$ .
<b>Resultados esperados</b>	El vehículo consigue adoptar las 2 referencias de vuelo indicadas. El vehículo se trasladará en la dirección negativa del eje $y$ .
<b>Parámetros de movimiento de interés</b>	Orientación en el eje $x$ : $\phi$ , referencia de <i>roll</i> : $\phi_D$ y posición en el eje $y$ : $y$ .
<b>Resultados</b>	 <p style="text-align: center;">Figura 44 - Resultados de la prueba 6</p>
<b>Comentarios</b>	Los errores en régimen estacionario son similares a los producidos anteriormente. En la primera referencia el error máximo producido es de $0.0154 \text{ rad} = 0.88^\circ$ . En la segunda referencia el error máximo producido es de $0.0868 \text{ rad} = 4.97^\circ$ .  La trayectoria es la esperada.



## 5.5.9 Prueba 7: Roll dinámico

<b>Descripción</b>	Se establece una referencia de roll que varía linealmente en función del tiempo.
<b>Referencias de vuelo</b>	$Throttle = 1500$ , $\phi_D(t) = 2.5t^\circ$ , $\theta_D = 0$ y $r_D = 0^\circ$ .
<b>Resultados esperados</b>	El vehículo adopta la referencia de vuelo establecida, trasladándose en la dirección negativa del eje $y$ .
<b>Parámetros de movimiento de interés</b>	Orientación en el eje $x$ : $\phi$ , referencia de roll: $\phi_D$ y posición en el eje $y$ : $y$ .
<b>Resultados</b>	 <p style="text-align: center;">Figura 45 - Resultados de la prueba 7</p>
<b>Comentarios</b>	<p>Se producen errores de magnitud variable en la orientación a lo largo de toda la simulación. El error máximo producido es de <math>0.0078 \text{ rad} = 0.4469^\circ</math>.</p> <p>La trayectoria es la esperada.</p>

## 6 Reflexión sobre los aspectos sociales, ambientales, éticos y profesionales

El impacto producido a estos aspectos por parte del proyecto se puede considerar nulo; el simulador no contribuye a la mejora de ningún Objetivo de Desarrollo Sostenible (ODS) ni constituye en principio ningún reto de cara a los aspectos profesionales.

## 7 Líneas de mejora

A continuación, presentamos las líneas de mejora destacadas:

- El simulador usa los ángulos de Euler para representar la orientación. Esta representación es válida para nuestra situación ya que no permitimos unas referencias de vuelo demasiado agresivas. Se podría representar la orientación usando *cuaternios* para extender los rangos de las referencias de vuelo.
- Se podría medir la velocidad de giro de un motor real de manera que la velocidad angular de los motores se represente de manera más fiel a la realidad.
- Se podría añadir la funcionalidad de añadir ruido a los sensores, de manera que se pueda evaluar el impacto de esta acción en la respuesta del controlador de vuelo.
- Se podría añadir un filtro de paso bajo al camino derivativo del controlador *PID*. Esto se debe a que en una configuración real el sensor tendrá ruido, lo que haría que el controlador *PID* implementado trabaje peor.
- Los paquetes *software* deberían ser implementados mediante parametrizaciones que incluyan interfaces en sus definiciones. De manera que se puedan representar los componentes de más modelos de multicopteros.
- Se podría implementar un visualizador en tiempo real usando *OpenGL* con *Ada*. Otra alternativa sería enviar los datos de simulación a través de *sockets*, de manera que un programa externo los procesase para su visualización, por ejemplo, usando la plataforma *Unity*.

## 8 Conclusiones

Se ha implementado satisfactoriamente el prototipo del simulador cumpliendo todos los objetivos propuestos. En particular, cabe destacar que todas las pruebas realizadas han resultado positivas, mostrando que se pueden realizar maniobras básicas de vuelo.

Adicionalmente, este prototipo sirve como punto de partida para la ampliación hacia un simulador más complejo. Por otra parte, a partir de este prototipo sería posible adaptar los componentes del controlador de vuelo, realizando pequeñas modificaciones, a un cuadricóptero real.

## 9 Anexos

### 9.1 Tests de planificabilidad de la prueba 3

Los tiempos de cómputo de las tareas se asignan midiendo el *worst-case execution time* (WCET) de cada una de ellas. Estos tiempos se han medido experimentalmente durante varias ejecuciones de la prueba 3. En la tabla siguiente resumimos los parámetros relevantes de las tareas para hacer los tests de planificabilidad.

Tarea	Periodo ( $T_i$ ) [ms]	WCET ( $C_i$ ) [ms]
Motors_Updater	0.1	0.022
Motion_Updater	5	1.863
Control_Algorithm	10	0.098
Absolute_Orientation_Sensor_Updater	10	0.140
Logger	50	0.112
Driver	60	0

Tabla 8 - Asignación de parámetros

Utilizamos 2 tests de planificabilidad simples mostrados en [1]. Estos tests son positivos si se cumple que:

$$\sum_{i=1}^N \frac{C_i}{T_i} \leq N(2^{1/N} - 1)$$

$$\prod_{i=1}^N \left( \frac{C_i}{T_i} + 1 \right) \leq 2$$

En nuestro caso, los resultados son los siguientes:

$$0.619 \leq 0.735$$

$$1.718 \leq 2$$

Por lo tanto, en principio, y a falta de un análisis más riguroso, el sistema sería planificable. Estos valores se han calculado con un *script* de MATLAB, el cual se puede encontrar en las siguientes páginas.

## 9.2 Script para ejecutar los tests de planificabilidad de la prueba 3

```
% utilization_tests.m
```

```
N = 6;
```

```
C = [0.022, 1.863, 0.098, 0.140, 0.112, 0.000];
```

```
T = [ 0.1,    5,    10,    10,    50,    60];
```

```
% Test 1.
```

```
sum = 0;
```

```
for i = 1:N
    sum = sum + (C(i) / T(i));
end
```

```
bound_1 = N * (2^(1 / N) - 1);
```

```
if sum <= bound_1
    fprintf('Test 1 passed: %f <= %f\n', sum, bound_1)
else
    fprintf('Test 1 failed: %f > %f\n', sum, bound_1)
end
```

```
% Test 2.
```

```
product = 1;
```

```
for i = 1:N
    product = product * (C(i) / T(i) + 1);
end
```

```
bound_2 = 2;
```

```
if product <= bound_2
    fprintf('Test 2 passed: %f <= %f\n', product, bound_2)
else
    fprintf('Test 2 failed: %f > %f\n', product, bound_2)
end
```

### 9.3 Cálculo del tensor de inercia

Para construir un tensor de inercia simple consideramos sólo aquellos componentes de mayor peso que nuestro cuadricóptero podría tener a la hora de la construcción física del mismo. Estos elementos son la batería, los 4 motores y los 4 brazos.

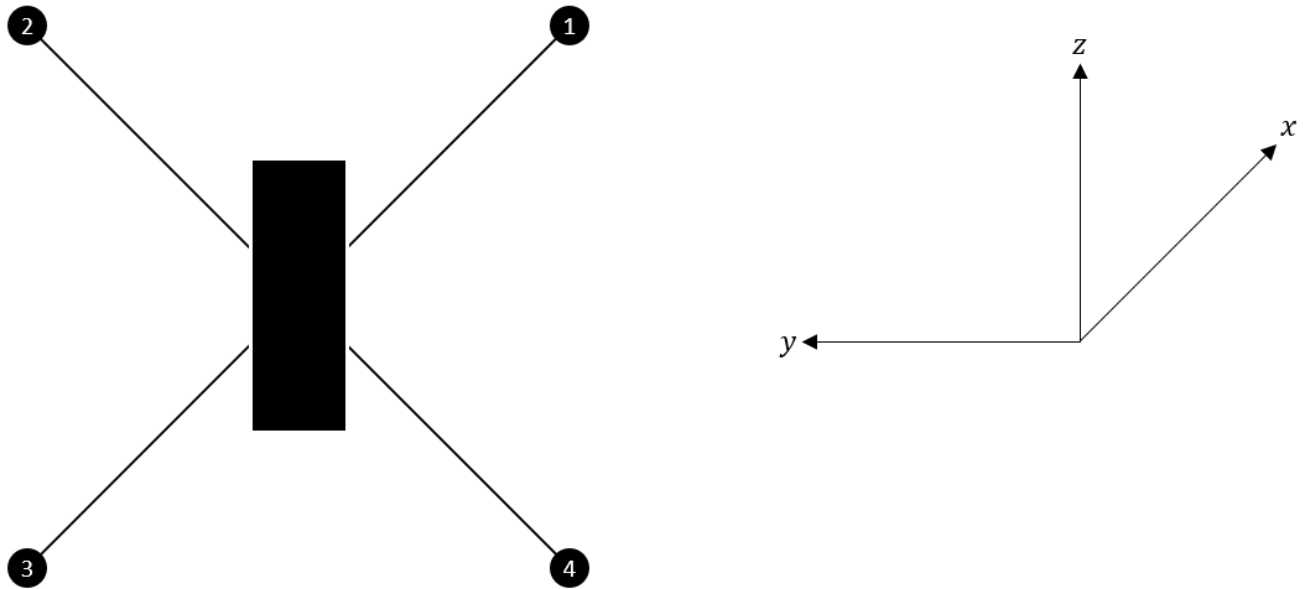


Figura 46 - Cálculo del tensor de inercia

Al tener acceso a estos componentes reales, vamos a usar sus medidas y pesos. La batería mide  $0.125 \times 0.042 \times 0.0325 \text{ m}$  (respectivamente en los ejes  $x$ ,  $y$  y  $z$ ) y pesa  $0.4 \text{ kg}$ , los motores tienen un radio de  $0.013 \text{ m}$  y una altura de  $0.026 \text{ m}$ , pesando  $0.056 \text{ kg}$ . Los brazos pesan  $0.056 \text{ kg}$ .

Las dimensiones de los brazos no son relevantes puesto que para evitar rotar sus tensores sólo vamos a usar su peso distribuyéndolo uniformemente entre los demás componentes. De esta manera, la nueva masa de la batería sería de  $0.4 + 2 \cdot 0.056 = 0.512 \text{ kg}$  (se añade a la batería el peso de dos brazos) y la nueva masa de un motor sería de  $0.056 + 0.056/2 = 0.084 \text{ kg}$  (a cada motor se le añade la mitad del peso de un brazo).

Para el cálculo del tensor de la inercia de la batería podemos considerar que es un *cuboide*, mientras que para el cálculo del tensor de inercia del motor podemos considerar que es un cilindro. Las expresiones usadas para calcular estos tensores de inercia aparecen en [11].

Una vez calculados estos dos tensores, para obtener el tensor final, basta con sumar al tensor de la batería el tensor desplazado de los 4 motores. Para el cálculo de estos tensores desplazados se usa el Teorema de *Steiner* [12]. A continuación, se puede ver el código en *MATLAB* empleado para automatizar este cálculo. Recordamos que la distancia desde el centro del vehículo hasta cualquiera de los motores es de  $0.225 \text{ m}$ .

```

% inertia.m

% Medidas y pesos.

battery_x = 0.125;
battery_y = 0.042;
battery_z = 0.0325;
battery_mass = 0.4;

motor_radius = 0.013;
motor_height = 0.026;
motor_mass = 0.056;

arm_mass = 0.056;

% Cálculo de la masa total.

total_mass = battery_mass + 4 * motor_mass + 4 * arm_mass;

% Tensor de la batería sumando a su masa la mitad del peso total de los 4
% brazos (2 brazos enteros en peso).

battery_additional_mass = 2 * arm_mass;

battery_i_x = ((battery_mass + battery_additional_mass) / 12) * (battery_y^2 + battery_z^2);
battery_i_y = ((battery_mass + battery_additional_mass) / 12) * (battery_x^2 + battery_z^2);
battery_i_z = ((battery_mass + battery_additional_mass) / 12) * (battery_x^2 + battery_y^2);

% Tensor de un motor sumando a su masa la mitad del peso de 1 brazo.

motor_additional_mass = arm_mass / 2;

motor_i_x = (1 / 12) * (motor_mass + motor_additional_mass) * (3 * motor_radius^2 + motor_height^2);
motor_i_y = (1 / 12) * (motor_mass + motor_additional_mass) * (3 * motor_radius^2 + motor_height^2);
motor_i_z = (1 / 2) * (motor_mass + motor_additional_mass) * motor_radius^2;

% Tensor desplazado de un motor sumando a su masa la mitad del peso de 1 brazo.

distance = 0.225;

motor_i_x_new = motor_i_x + ((motor_mass + motor_additional_mass) * ((sqrt(2)/2) * distance)^2);
motor_i_y_new = motor_i_y + ((motor_mass + motor_additional_mass) * ((sqrt(2)/2) * distance)^2);
motor_i_z_new = motor_i_z;

% Tensor total.

total_i_x = battery_i_x + 4 * motor_i_x_new;
total_i_y = battery_i_y + 4 * motor_i_y_new;
total_i_z = battery_i_z + 4 * motor_i_z_new;

```



## 9.4 Código de las pruebas

### 9.4.1 Prueba 1

En *configuration.ads*:

```
-- ...

File_Path : aliased String := "./test_1.csv";

-- ...
```

En *driver.adb*:

```
Throttle_Reference : constant Float := 1000.0;
Roll_Reference : constant Float := Utils.Degrees_To_Radians (0.0);
Pitch_Reference : constant Float := Utils.Degrees_To_Radians (0.0);
Yaw_Rate_Reference : constant Float := Utils.Degrees_To_Radians (0.0);
```

En *plot\_test\_1.m*:

```
% plot_test_1.m

data = csvread('...', 1, 0);

tiledlayout(3, 1)

% z
nexttile
plot(data(:, 1), data(:, 16))
title('Fuerza en el eje z')
xlabel('Time [s]')
ylabel('Z [N]')

% w
nexttile
plot(data(:, 1), data(:, 22))
title('Velocidad lineal en el eje z')
xlabel('Time [s]')
ylabel('w [m/s]')

% z
nexttile
plot(data(:, 1), data(:, 28))
title('Posición en el eje z')
xlabel('Time [s]')
ylabel('z [m]')
```

### 9.4.2 Prueba 2

En *configuration.ads*:

```
-- ...

File_Path : aliased String := "./test_2.csv";

-- ...
```

En *driver.adb*:

```
Throttle_Reference : constant Float := 2000.0;
Roll_Reference : constant Float := Utils.Degrees_To_Radians (0.0);
Pitch_Reference : constant Float := Utils.Degrees_To_Radians (0.0);
Yaw_Rate_Reference : constant Float := Utils.Degrees_To_Radians (0.0);
```

En *plot\_test\_2.m*:

```
% plot_test_2.m

data = csvread('...', 1, 0);

tiledlayout(3, 1)

% Z
nexttile
plot(data(:, 1), data(:, 16))
title('Fuerza en el eje z')
xlabel('Time [s]')
ylabel('Z [N]')

% w
nexttile
plot(data(:, 1), data(:, 22))
title('Velocidad lineal en el eje z')
xlabel('Time [s]')
ylabel('w [m/s]')

% z
nexttile
plot(data(:, 1), data(:, 28))
title('Posición en el eje z')
xlabel('Time [s]')
ylabel('z [m]')
```

### 9.4.3 Prueba 3

En *configuration.ads*:

```
-- ...

File_Path : aliased String := "./test_3.csv";

-- ...
```

En *driver.adb*:

```
Throttle_Reference : constant Float := 1500.0;
Roll_Reference : constant Float := Utils.Degrees_To_Radians (25.0);
Pitch_Reference : constant Float := Utils.Degrees_To_Radians (0.0);
Yaw_Rate_Reference : constant Float := Utils.Degrees_To_Radians (0.0);
```

En *plot\_test\_3.m*:

```
% plot_test_3.m

data = csvread('...', 1, 0);

tiledlayout(2, 1)

% phi
nexttile
plot(data(:, 1), data(:, 29), data(:, 1), data(:, 3))
title('Orientación en el eje x')
xlabel('Time [s]')
ylabel('\phi [rad]')
legend({'Real', 'Reference'}, 'Location', 'southeast')

% y
nexttile
plot(data(:, 1), data(:, 27))
title('Posición en el eje y')
xlabel('Time [s]')
ylabel('y [m]')
```

#### 9.4.4 Prueba 4

En *configuration.ads*:

```
-- ...

File_Path : aliased String := "./test_4.csv";

-- ...
```

En *driver.adb*:

```
Throttle_Reference : constant Float := 1500.0;
Roll_Reference : constant Float := Utils.Degrees_To_Radians (0.0);
Pitch_Reference : constant Float := Utils.Degrees_To_Radians (25.0);
Yaw_Rate_Reference : constant Float := Utils.Degrees_To_Radians (0.0);
```

En *plot\_test\_4.m*:

```
% plot_test_4.m

data = csvread('...', 1, 0);

tiledlayout(2, 1)

% theta
nexttile
plot(data(:, 1), data(:, 30), data(:, 1), data(:, 4))
title('Orientación en el eje y')
xlabel('Time [s]')
ylabel('\theta [rad]')
legend({'Real', 'Reference'}, 'Location', 'southeast')

% x
nexttile
plot(data(:, 1), data(:, 26))
title('Posición en el eje x')
xlabel('Time [s]')
ylabel('x [m]')
```

#### 9.4.5 Prueba 5

En *configuration.ads*:

```
-- ...

File_Path : aliased String := "./test_5.csv";

-- ...
```

En *driver.adb*:

```
Throttle_Reference : constant Float := 1500.0;
Roll_Reference : constant Float := Utils.Degrees_To_Radians (0.0);
Pitch_Reference : constant Float := Utils.Degrees_To_Radians (0.0);
Yaw_Rate_Reference : constant Float := Utils.Degrees_To_Radians (180.0);
```

En *plot\_test\_5.m*:

```
% plot_test_5.m

data = csvread('...', 1, 0);

tiledlayout(2, 1)

% r
nexttile
plot(data(:, 1), data(:, 25), data(:, 1), data(:, 5))
title('Velocidad angular en el eje z')
xlabel('Time [s]')
ylabel('r [rad/s]')
legend({'Real', 'Reference'}, 'Location', 'southeast')

% psi
nexttile
plot(data(:, 1), data(:, 31))
title('Orientación en el eje z')
xlabel('Time [s]')
ylabel('\psi [rad]')
```

### 9.4.6 Prueba 6

En *configuration.ads*:

```
-- ...

File_Path : aliased String := "./test_6.csv";

-- ...
```

En *driver.adb*:

```
-----
-- On_Start --
-----

procedure On_Start (Self : in out Bundle) is

  Throttle_Reference : constant Float := 1500.0;
  Roll_Reference : constant Float := Utils.Degrees_To_Radians (12.5);
  Pitch_Reference : constant Float := Utils.Degrees_To_Radians (0.0);
  Yaw_Rate_Reference : constant Float := Utils.Degrees_To_Radians (0.0);

begin

  Self.References.Update (Throttle_Reference, Roll_Reference, Pitch_Reference, Yaw_Rate_Reference);

end On_Start;

-----
-- On_Loop --
-----

procedure On_Loop (Self : in out Bundle; Elapsed_Time : Time_Span) is

  Throttle_Reference : constant Float := 1500.0;
  Roll_Reference : constant Float := Utils.Degrees_To_Radians (25.0);
  Pitch_Reference : constant Float := Utils.Degrees_To_Radians (0.0);
  Yaw_Rate_Reference : constant Float := Utils.Degrees_To_Radians (0.0);

begin

  if Elapsed_Time >= Seconds (5) then

    Self.References.Update (Throttle_Reference, Roll_Reference, Pitch_Reference,
    Yaw_Rate_Reference);

    end if;

end On_Loop;
```

En *plot\_test\_6.m*:

```
% plot_test_6.m

data = csvread('...', 1, 0);

tiledlayout(2, 1)

% phi
nexttile
plot(data(:, 1), data(:, 29), data(:, 1), data(:, 3))
title('Orientación en el eje x')
xlabel('Time [s]')
ylabel('\phi [rad]')
legend({'Real', 'Reference'}, 'Location', 'southeast')

% y
nexttile
plot(data(:, 1), data(:, 27))
title('Posición en el eje y')
xlabel('Time [s]')
ylabel('y [m]')
```

### 9.4.7 Prueba 7

En *configuration.ads*:

```
-- ...

File_Path : aliased String := "./test_7.csv";

-- ...
```

En *driver.adb*:

```
-----
-- On_Start --
-----

procedure On_Start (Self : in out Bundle) is
begin
  null;
end On_Start;

-----
-- On_Loop --
-----

procedure On_Loop (Self : in out Bundle; Elapsed_Time : Time_Span) is

  Throttle_Reference : constant Float := 1500.0;
  Roll_Reference : Float;
  Pitch_Reference : constant Float := Utils.Degrees_To_Radians (0.0);
  Yaw_Rate_Reference : constant Float := Utils.Degrees_To_Radians (0.0);

begin

  Roll_Reference := Utils.Degrees_To_Radians (2.5 * Float (Elapsed_Time / Milliseconds (1)) * 0.001);

  Self.References.Update (Throttle_Reference, Roll_Reference, Pitch_Reference, Yaw_Rate_Reference);

end On_Loop;
```

En *plot\_test\_7.m*:

```
% plot_test_7.m

data = csvread('...', 1, 0);

tiledlayout(2, 1)

% phi
nexttile
plot(data(:, 1), data(:, 29), data(:, 1), data(:, 3))
title('Orientación en el eje x')
xlabel('Time [s]')
ylabel('\phi [rad]')
legend({'Real', 'Reference'}, 'Location', 'southeast')

% y
nexttile
plot(data(:, 1), data(:, 27))
title('Posición en el eje y')
xlabel('Time [s]')
ylabel('y [m]')
```



## 10 Bibliografía

- [1] A. Burns y A. Wellings, *Analysable Real-Time Systems: Programmed in Ada*, University of York, 2016.
- [2] «Ravenscar profile,» [En línea]. Available: [https://en.wikipedia.org/wiki/Ravenscar\\_profile](https://en.wikipedia.org/wiki/Ravenscar_profile).
- [3] T. I. Fossen, *Guidance and Control of Ocean Vehicles*, Wiley, 1994.
- [4] A. Tayebi y S. McGilvray, «Attitude stabilization of a four-rotor aerial robot,» *43rd IEEE Conference on Decision and Control*, 2004.
- [5] «Euler method,» [En línea]. Available: [https://en.wikipedia.org/wiki/Euler\\_method](https://en.wikipedia.org/wiki/Euler_method).
- [6] «Linear interpolation,» [En línea]. Available: [https://en.wikipedia.org/wiki/Linear\\_interpolation](https://en.wikipedia.org/wiki/Linear_interpolation).
- [7] N. S. Nise, *Control Systems Engineering*, 8th ed., Wiley, 2019.
- [8] «Trapezoidal rule,» [En línea]. Available: [https://en.wikipedia.org/wiki/Trapezoidal\\_rule](https://en.wikipedia.org/wiki/Trapezoidal_rule).
- [9] D. Wilson, «Teaching Your PI Controller to Behave (Part VII),» 13 04 2013. [En línea]. Available: [https://e2e.ti.com/blogs\\_/b/industrial\\_strength/archive/2013/04/13/teaching-your-pi-controller-to-behave-part-vii](https://e2e.ti.com/blogs_/b/industrial_strength/archive/2013/04/13/teaching-your-pi-controller-to-behave-part-vii). [Último acceso: 13 07 2020].
- [10] «How To Choose Motor For Racing Drone & Quadcopter,» [En línea]. Available: <https://oscarliang.com/quadcopter-motor-propeller/>.
- [11] «List of moments of inertia,» [En línea]. Available: [https://en.wikipedia.org/wiki/List\\_of\\_moments\\_of\\_inertia](https://en.wikipedia.org/wiki/List_of_moments_of_inertia).
- [12] «Parallel axis theorem,» [En línea]. Available: [https://en.wikipedia.org/wiki/Parallel\\_axis\\_theorem](https://en.wikipedia.org/wiki/Parallel_axis_theorem).