



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



TRABAJO FIN DE GRADO

Diseño de un sistema de simulación de misiones
autónomas para sistemas aéreos no tripulados.
Aplicación al proyecto HORUS UPV

AUTOR

GARCÍA CASES, Arnau argarca@etsid.upv.es

TUTOR

GARCÍA-NIETO RODRÍGUEZ, Sergio sgnieto@isa.upv.es

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO

GRADO EN INGENIERÍA AEROESPACIAL
CURSO 2020-2021

En este archivo se incluyen los siguientes documentos ¹:

MEMORIA

PLIEGO DE CONDICIONES

PRESUPUESTO

MANUAL DE USUARIO

¹Todas las figuras utilizadas en estos documentos que no contienen referencias son de elaboración propia.



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



TRABAJO FIN DE GRADO

Diseño de un sistema de simulación de misiones
autónomas para sistemas aéreos no tripulados.
Aplicación al proyecto HORUS UPV

Memoria

AUTOR

GARCÍA CASES, Arnau argarca@etsid.upv.es

TUTOR

GARCÍA-NIETO RODRÍGUEZ, Sergio sgnieto@isa.upv.es

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO

GRADO EN INGENIERÍA AEROESPACIAL
CURSO 2020-2021

Resumen

El presente trabajo desarrolla un sistema de simulación de misiones autónomas, excluyendo las fases de despegue y aterrizaje, que permite visualizar y verificar la respuesta de sistemas aéreos no tripulados a lo largo de la misma. Este sistema se construye a partir del modelo dinámico y sistema de control de la aeronave del proyecto HORUS UPV, realizados en trabajos previos. La definición de las misiones autónomas se realiza mediante una estación de control terrestre que se comunica con el sistema de simulación. En el mismo se incluye un entorno de visualización creado mediante la plataforma de creación 3D Unreal Engine, a partir de un Modelo Digital del Terreno (MDT) y una imagen satelital, en el cual se visualiza el estado de la aeronave en todo momento. Además, a lo largo de la misión se extraen datos de este entorno en tiempo real, para ser analizados por un algoritmo de visión artificial basado en aprendizaje automático, el cual detecta una serie de balizas incluidas en el mismo. Finalmente, las coordenadas de las balizas son calculadas a partir de las detecciones del algoritmo de visión artificial, obteniendo errores absolutos en la posición inferiores a 2.5 m.

Abstract

This project develops a simulation system for autonomous missions, excluding take-off and landing phases, which allows the visualization and verification of the response of unmanned aerial systems throughout the mission. This system uses an existing dynamic model and control system of the HORUS UPV aircraft, developed in previous works. The autonomous mission plan is carried out by means of a ground control station that communicates with the simulation system. The simulation system includes a realistic environment in which the status of the aircraft is displayed at all times, created with Unreal Engine, a 3D creation platform, based on a Digital Terrain Model (DTM) and a satellite image. In addition, throughout the mission, data is extracted from this environment in real time, so that a computer vision algorithm, based on machine learning, detects a series of ground markers included in it. Finally, the marker's coordinates are calculated from the location of the markers in each video frame, identified by the computer vision algorithm, obtaining absolute position errors smaller than 2.5 m.

Resum

El present treball desenvolupa un sistema de simulació de missions autònomes, exclouent les fases d'enlairament i aterratge, que permet visualitzar i verificar la resposta de sistemes aeris no tripulats al llarg d'aquesta. Aquest sistema es construeix a partir del model dinàmic i sistema de control de l'aronau del projecte HORUS UPV, realitzats en treballs previs. La definició de les missions autònomes es realitza mitjançant una estació de control terrestre que es comunica amb el sistema de simulació. En el mateix s'inclou un entorn de visualització creat mitjançant la plataforma de creació 3D Unreal Engine, a partir d'un Model Digital del Terreny (MDT) i una imatge satel·litària, en el qual es visualitza l'estat de l'aeronau en tot moment. A més, al llarg de la missió s'extrauen dades d'aquest entorn en temps real, per a ser analitzats per un algorisme de visió artificial basat en aprenentatge automàtic, el qual detecta una sèrie de balises incloses en aquest. Finalment, les coordenades de les balises són calculades a partir de les deteccions de l'algorisme de visió artificial, obtenint errors absoluts en la posició inferiors a 2.5 m.

Índice general

Índice de figuras	II
Índice de cuadros	IV
Lista de símbolos	VII
1 Introducción y antecedentes	1
1.1 Introducción	1
1.2 Aeronaves no tripuladas	1
1.2.1 Clasificación	2
1.2.2 Aplicaciones	3
2 Objeto y alcance del proyecto	5
2.1 Objeto del proyecto	5
2.2 Alcance del proyecto	5
3 Estudio de necesidades, factores a considerar: limitaciones y condicionantes	7
3.1 Normativa del concurso <i>UAS Challenge</i>	7
3.1.1 Requisitos operacionales y de diseño	7
3.1.2 Misión	9
4 Planteamiento de soluciones alternativas y justificación de la solución	11
4.1 Simulación de sistemas dinámicos	11
4.2 Estrategias de control	11
4.3 Entorno de simulación	14
4.4 Estación de control terrestre	15
4.5 Algoritmos de Visión Artificial	15
5 Plataforma de vuelo	17
5.1 Modelado	17
5.1.1 Sistemas de referencia y actitud de la aeronave	17
5.1.2 Ecuaciones de <i>Bryan</i>	18
5.1.3 Características de la plataforma	21
5.1.4 Modelo propulsivo	22
5.1.5 Fuerzas y momentos aerodinámicos	23
5.2 Sistema de control	24
5.2.1 Arquitectura	24
5.2.2 Linealización del modelo	25
5.2.3 Ajuste de los controladores	30

6 Descripción y justificación detallada de la solución adoptada	33
6.1 Implementación del sistema aéreo no tripulado	33
6.1.1 Modelo dinámico	34
6.1.2 Sistema de control	36
6.2 Estación de control terrestre	40
6.2.1 Definición de la misión	40
6.2.2 Conexión con <i>Simulink</i>	40
6.3 Entorno de simulación realista	41
6.3.1 Implementación en Simulink	41
6.3.2 Creación del escenario realista	42
6.4 Visión artificial	47
6.4.1 Balizas	47
6.4.2 Algoritmo de detección de balizas	48
6.4.3 Obtención de las coordenadas de las balizas	53
6.4.4 Implementación en <i>Simulink</i>	55
7 Resultados	57
7.1 Misión	57
7.2 Sistema de control	58
7.3 Visión artificial	63
8 Conclusiones y trabajos futuros	67
Bibliografía	70

Índice de figuras

4.1	Estructura de un controlador PID [1]	12
4.2	Estructura de un controlador basado en lógica difusa [2]	12
4.3	Sistema en lazo cerrado con LQR [3]	13
4.4	Estructura básica del MPC [4]	13
4.5	Esquema de control neuronal adaptativo [5]	14
5.1	Arquitectura del sistema de control	25
6.1	Esquema general de la implementación en Simulink	33
6.2	Esquema general del bloque <i>UAS</i>	34
6.3	Implementación del modelo dinámico	35
6.4	Implementación del cálculo de fuerzas y momentos totales	36
6.5	Implementación de los controladores del lazo de control interno	37
6.6	Implementación del bloque de guiado	38
6.7	Cálculo de las referencias utilizadas por el lazo de control interno	39
6.8	Misión definida en QGC	40
6.9	Conexión con QGC en Simulink	41
6.10	Comunicación con el entorno de simulación 3D [6]	42
6.11	Implementación de la comunicación con el entorno realista	43
6.12	Archivos necesarios para la creación del escenario realista	46
6.13	Resultado final del escenario en Unreal Engine	46
6.14	Balizas utilizadas en la simulación	48
6.15	Aplicación <i>Video Labeler</i>	50
6.16	Rendimiento del detector con los parámetros por defecto.	52
6.17	Modelo de cámara estenopeica. Recuperado de la documentación de <i>Computer Vision Toolbox</i> , de MATLAB®	53
6.18	Vector normalizado indicando la baliza	54
6.19	Implementación en <i>Simulink</i> de la detección de balizas	56
7.1	Visualización de la misión en QGC.	57
7.2	Representación de la trayectoria	58
7.3	Evolución temporal de la posición de la aeronave respecto a la trayectoria de referencia	59
7.4	Evolución temporal de las acciones de control	60
7.5	Evolución temporal de las referencias emitidas por el lazo de control externo	61
7.6	Evolución temporal de las velocidades en ejes cuerpo	61
7.7	Evolución temporal de las velocidades angulares	62
7.8	Evolución temporal del ángulo de ataque y derrape	62
7.9	Evolución temporal de las velocidades en el sistema de referencia NED	63
7.10	Visualización de la aeronave en el escenario realista	64

7.11 Visualización de la imagen captada por la cámara, con la identificación de la baliza 65
7.12 Errores en la determinación de la posición de las balizas 65

Índice de cuadros

1.1	Clasificación de los UAS en función de su alcance, altitud, autonomía y MTOW [7].	2
5.1	Nomenclatura utilizada en las ecuaciones de Bryan.	19
5.2	Propiedades geométricas de la aeronave.	21
5.3	Propiedades másicas e inerciales de la aeronave.	21
5.4	Derivadas aerodinámicas de las variables de estado	22
5.5	Derivadas aerodinámicas de las variables de control	22
5.6	Valor de las variables de estado en el punto de funcionamiento	26
5.7	Polos y ceros de la función de transferencia ϕ/δ_A	28
5.8	Polos y ceros de la función de transferencia θ/δ_E	28
5.9	Polos y ceros de la función de transferencia u/δ_P	29
5.10	Polos y ceros de la función de transferencia \dot{v}/δ_R	29
5.11	Polos y ceros de la función de transferencia z/θ	30
5.12	Polos y ceros de la función de transferencia ζ/ϕ	30
5.13	Limitaciones de las acciones de control	31
5.14	Ganancias de los controladores del lazo interno	31
5.15	Ganancias de los controladores del lazo externo	32
6.1	Tamaños del mapa recomendados para Unreal Engine [8]	45
6.2	Comparativa del rendimiento entre los detectores entrenados	52
6.3	Parámetros de la cámara utilizada en la simulación	53
7.1	Waypoints de la misión de simulación	58

Lista de símbolos

h	Altura
ϕ	Ángulo de alabeo
α	Ángulo de ataque
θ	Ángulo de cabeceo
β	Ángulo de derrape
ψ	Ángulo de guiñada
C_X	Coefficiente de fuerza en dirección X
C_Z	Coefficiente de fuerza en dirección Z
C_Y	Coefficiente de fuerza lateral
C_l	Coefficiente de momento de alabeo
C_M	Coefficiente de momento de cabeceo
C_{M_0}	Coefficiente de momento de cabeceo con ángulo de ataque nulo
C_N	Coefficiente de momento de guiñada
C_D	Coefficiente de resistencia
C_L	Coefficiente de sustentación
C_{L_0}	Coefficiente de sustentación con ángulo de ataque nulo
x	Coordenada x del sistema NED (norte)
y	Coordenada y del sistema NED (este)
z	Coordenada z del sistema NED (abajo)
c_w	Cuerda media aerodinámica
δ_A	Deflexión de los alerones
δ_E	Deflexión de los elevadores (estabilizador horizontal)
δ_R	Deflexión del timón de dirección
ρ	Densidad atmosférica
$C_{Y_{\delta_A}}$	Derivada del coeficiente de fuerza lateral respecto a la deflexión de los alerones

$C_{Y_{\delta R}}$	Derivada del coeficiente de fuerza lateral respecto a la deflexión del timón de dirección
$C_{Y_{\beta \text{Dot}}}$	Derivada del coeficiente de fuerza lateral respecto a la variación temporal del ángulo de derrape
C_{Y_p}	Derivada del coeficiente de fuerza lateral respecto a la velocidad de alabeo
C_{Y_r}	Derivada del coeficiente de fuerza lateral respecto a la velocidad de guiñada
$C_{Y_{\beta}}$	Derivada del coeficiente de fuerza lateral respecto al ángulo de derrape
$C_{l_{\delta A}}$	Derivada del coeficiente de momento de alabeo respecto a la deflexión de los alerones
$C_{l_{\delta R}}$	Derivada del coeficiente de momento de alabeo respecto a la deflexión del timón de dirección
$C_{l_{\beta \text{Dot}}}$	Derivada del coeficiente de momento de alabeo respecto a la variación del ángulo de derrape
C_{l_p}	Derivada del coeficiente de momento de alabeo respecto a la velocidad de alabeo
C_{l_r}	Derivada del coeficiente de momento de alabeo respecto a la velocidad de guiñada
$C_{l_{\beta}}$	Derivada del coeficiente de momento de alabeo respecto al ángulo de derrape
$C_{M_{\delta E}}$	Derivada del coeficiente de momento de cabeceo respecto a la deflexión de los elevadores
$C_{M_{\alpha \text{Dot}}}$	Derivada del coeficiente de momento de cabeceo respecto a la variación temporal del ángulo de ataque
C_{M_q}	Derivada del coeficiente de momento de cabeceo respecto a la velocidad de cabeceo
$C_{M_{\alpha}}$	Derivada del coeficiente de momento de cabeceo respecto al ángulo de ataque
$C_{N_{\delta A}}$	Derivada del coeficiente de momento de guiñada respecto a la deflexión de los alerones
$C_{N_{\delta R}}$	Derivada del coeficiente de momento de guiñada respecto a la deflexión del timón de dirección
$C_{N_{\beta \text{Dot}}}$	Derivada del coeficiente de momento de guiñada respecto a la variación temporal del ángulo de derrape
C_{N_p}	Derivada del coeficiente de momento de guiñada respecto a la velocidad de alabeo
C_{N_r}	Derivada del coeficiente de momento de guiñada respecto a la velocidad de guiñada
$C_{N_{\beta}}$	Derivada del coeficiente de momento de guiñada respecto al ángulo de derrape
$C_{L_{\delta E}}$	Derivada del coeficiente de sustentación respecto a la deflexión de los elevadores
$C_{L_{\alpha \text{Dot}}}$	Derivada del coeficiente de sustentación respecto a la variación temporal del ángulo de ataque
C_{L_q}	Derivada del coeficiente de sustentación respecto a la velocidad de cabeceo
$C_{L_{\alpha}}$	Derivada del coeficiente de sustentación respecto al ángulo de ataque
Th	Empuje
b_w	Envergadura alar

F_{AX}	Fuerza aerodinámica frontal
F_{AY}	Fuerza aerodinámica lateral
F_{AZ}	Fuerza aerodinámica vertical
F	Fuerza neta
F_{TX}	Fuerza propulsiva frontal
F_{TY}	Fuerza propulsiva lateral
F_{TZ}	Fuerza propulsiva vertical
K_d	Ganancia derivativa del regulador
K_i	Ganancia integral del regulador
K_p	Ganancia proporcional del regulador
m	Masa de la aeronave
H	Momento angular
L	Momento de alabeo
M_{TX}	Momento de alabeo asociado a fuerzas propulsivas
M	Momento de cabeceo
M_{TY}	Momento de cabeceo asociado a fuerzas propulsivas
N	Momento de guiñada
M_{TZ}	Momento de guiñada asociado a fuerzas propulsivas
I_{xx}	Momento de inercia sobre el eje X
I_{yy}	Momento de inercia sobre el eje Y
I_{zz}	Momento de inercia sobre el eje Z
δ_P	Posición de la palanca de gases (control del empuje del motor)
p	Presión atmosférica
I_{xy}	Producto de inercia XY
I_{xz}	Producto de inercia XZ
I_{yx}	Producto de inercia YX
I_{yz}	Producto de inercia YZ
I_{zx}	Producto de inercia ZX
I_{zy}	Producto de inercia ZY
ζ	Rumbo
S_w	Superficie alar
T	Temperatura atmosférica
V	Velocidad aerodinámica
p	Velocidad angular de alabeo

q	Velocidad angular de cabeceo
r	Velocidad angular de guiñada
ω	Velocidad angular del triedo de ejes cuerpo respecto al sistema inercial
u	Velocidad lineal frontal en ejes cuerpo
v	Velocidad lineal lateral en ejes cuerpo
w	Velocidad lineal vertical en ejes cuerpo

1 | Introducción y antecedentes

1.1. Introducción

HORUS UPV, anteriormente denominado HERMES-UPV, es un proyecto multidisciplinar de carácter universitario, con el objetivo de diseñar, planificar y fabricar una aeronave de ala fija. Este proyecto se realiza en colaboración con el Grupo de Control Predictivo y Optimización Heurística (CPOH) del Instituto Universitario de Automática e Informática Industrial (ai2).

La motivación detrás de este proyecto reside en la presentación de la aeronave al concurso internacional *UAS Challenge*, organizado por el iMechE (*Institute of Mechanical Engineers*). Este concurso tiene como objetivo “desarrollar ingenieros profesionales e inspirar a las siguientes generaciones”. Para ello, equipos de estudiantes de todo el mundo forman parte del concurso anual, en el que se lleva a cabo el diseño completo y la construcción de un sistema aéreo no tripulado para realizar una misión de ayuda humanitaria. Esta misión pretende simular un escenario de desastre natural en el que se cortan todas las vías terrestres de comunicación con los afectados, de modo que sea necesario proveer de ayuda humanitaria por vía aérea, además de realizar búsquedas en el área afectada para identificar a los damnificados. De este modo, se proveería con la ayuda vital necesaria hasta que se desarrollara un plan de evacuación a gran escala.

Con el objetivo de llevar a cabo el desarrollo de un sistema aéreo no tripulado que cumpla con esta finalidad, el proyecto HORUS UPV está compuesto por varios grupos de trabajo, como son el grupo de diseño CAD, el grupo de cálculo estructural y aerodinámico, el de fabricación, el grupo de visión artificial o el grupo de sistemas y control de vuelo. Dentro de este último, el objetivo es desarrollar e implementar el sistema de control para cumplir con los requerimientos del concurso. Es en este grupo de trabajo en el que se desarrolla el presente Trabajo Fin de Grado (TFG).

1.2. Aeronaves no tripuladas

Según el marco normativo de la OACI (Organización de Aviación Civil Internacional), citando textualmente, establece que “un vehículo aéreo no tripulado es una aeronave sin piloto en el sentido del Artículo 8 del Convenio sobre Aviación Civil Internacional, que vuela sin un piloto al mando a bordo y que se controla a distancia y plenamente desde otro lugar (tierra, otra aeronave, espacio) o ha sido programada y es plenamente autónoma” [9]. En esta normativa también se definen los términos utilizados para referirse a aeronaves no tripuladas¹:

¹En términos coloquiales se utiliza la palabra dron para referirse de forma genérica a aeronaves no tripuladas, sin distinguir entre aquellas pilotadas por control remoto o aquellas completamente autónomas.

- UA - Aeronave no tripulada: aeronave destinada a volar sin piloto.
- UAS - Sistema(s) de aeronave(s) no tripulada(s): aeronave y sus elementos conexos que operan sin piloto a bordo.
- RPA - Aeronave pilotada a distancia: aeronave que no lleva a bordo un piloto a los mandos. Es una subcategoría de las aeronaves no tripuladas.
- RPAS - Sistema de aeronave pilotada a distancia: conjunto de elementos configurables integrado por una aeronave pilotada a distancia, sus estaciones de piloto remoto conexas, los necesarios enlaces de mando y control y cualquier otro elemento de sistema que pueda requerirse en cualquier punto durante la operación de vuelo.
- UAV - Vehículo aéreo no tripulado: término obsoleto según el marco normativo de la OACI. En la segunda reunión oficial de la OACI (Palm Coast, Florida, EUA, enero de 2007) se sugirió que el objeto temático pasara a denominarse sistemas de aeronaves no tripuladas (UAS).

1.2.1. Clasificación

La clasificación de los UAS puede realizarse atendiendo a distintos criterios. Una de ellas es atendiendo a sus capacidades de vuelo, en base al alcance, la altitud, la autonomía y el peso máximo al despegue, MTOW (*Maximum Take-Off Weight*). La Tabla 1.1 representa la clasificación de las aeronaves no tripuladas de acuerdo a estos parámetros.

Categoría de UAS	Alcance (km)	Altitud (m)	Autonomía (h)	MTOW (kg)
Estratosféricos	> 2000	20000 – 30000	48	< 3000
Elevada altitud y gran autonomía (HALE)	> 2000	20000	48	15000
Altitud media y gran autonomía (MALE)	> 500	14000	24 – 48	1500
Baja altitud y gran autonomía (LALE)	> 500	3000	≈ 24	≈ 30
Baja altitud y penetración profunda	> 250	50 – 9000	0.25 - 1	350
Alcance medio	70 a > 500	8000	6 – 18	1250
Alcance corto	10 – 70	3000	3 – 6	200
Mini	< 10	< 300	< 2	< 30
Micro	< 10	< 250	< 0.5	< 1

Tabla 1.1. Clasificación de los UAS en función de su alcance, altitud, autonomía y MTOW [7].

Otra clasificación corresponde con el tipo de ala, en concreto:

- UA de ala fija: este tipo de aeronaves suelen tener un despegue horizontal, y según la geometría de las alas se pueden encontrar diferentes diseños: ala volante, delta, convencional, “*canard*”, “*blended-body-wing*”, “*joined wing*”, rectangular o trapezoidal, con o sin flecha, ala alta, media o baja, etcétera. También se pueden clasificar de acuerdo a la geometría de la cola, ya sea un estabilizador horizontal de cola, cola en V, doble cola o en forma de H.
- UA de ala rotatoria: estas aeronaves vuelan aprovechando la sustentación que generan sus alas rotatorias o palas del motor. Se suelen clasificar de acuerdo al número de rotores, que oscilan entre tres (tricopters) y ocho (octocopters), siendo la configuración de cuatro motores (quadrotor) la más común.

La aeronave desarrollada en el proyecto HORUS UPV corresponde con una aeronave de ala fija rectangular, sin flecha, ala alta, y estabilizador horizontal de cola.

1.2.2. Aplicaciones

Las aplicaciones de las aeronaves no tripuladas son de diversa índole y con propósitos distintos, entre ellas destacan:

- Búsqueda de personas desaparecidas: facilitan la búsqueda en terrenos de difícil acceso, como zonas montañosas o nevadas. Estas permiten realizar las labores de búsqueda de forma ininterrumpida, debido a su reducido tamaño, la facilidad para la puesta en marcha y el bajo coste en comparación con un helicóptero tradicional.
- Fotografía, vídeo y cartografía aérea: desde la realización de vídeos y fotografías en el ámbito comercial, hasta labores de investigación.
- Monitorización de carreteras: control de tráfico y detección de infracciones.
- Monitorización de recursos naturales: como puede ser la prevención y control de incendios forestales.
- Entretenimiento: para actividades de ocio o deportivas, como las carreras de drones.
- Fotos y vídeo: utilizadas en el ámbito tanto personal como profesional, para la realización de tomas aéreas.
- Protección de la vida silvestre: monitorización de la caza para detectar actividades ilegales, y monitorización de fauna y flora, entre otros.
- Auxilio en catástrofes: entrega de ayuda humanitaria en zonas con dificultad de acceso tras una catástrofe, como puede ser un tsunami o sismo.

En general, el uso de aeronaves no tripuladas ha crecido de manera exponencial durante los últimos años, debido a sus capacidades frente a vehículos aéreos tradicionales, como helicópteros. No solo evitan poner en riesgo vidas humanas, también reducen significativamente los costes al tener un peso y consumo de combustible o energía menor.

2 | Objeto y alcance del proyecto

2.1. Objeto del proyecto

El objetivo principal de este proyecto corresponde con el desarrollo de un sistema de simulación para evaluar el comportamiento de la aeronave diseñada en el proyecto HORUS UPV, durante misiones autónomas que comprenden las fases de vuelo de ascenso, crucero y descenso, con la capacidad de modificar los escenarios de simulación, los parámetros del sistema de guiado y control, y la inclusión de algoritmos de visión artificial para la detección de objetos en el escenario.

2.2. Alcance del proyecto

El objetivo de este proyecto se conseguirá a través de una serie de hitos:

1. Adaptar la estructura del sistema de control y modelo dinámico de la aeronave diseñada en el proyecto HORUS UPV, realizados en trabajos previos, a la estructura del sistema de simulación desarrollado en este proyecto.
2. Establecer una conexión mediante el sistema de simulación y una estación de control terrestre para definir las misiones autónomas.
3. Crear un escenario que sirva como base del entorno de simulación.
4. Visualizar, en el escenario, el estado de la aeronave durante la misión.
5. Implementar algoritmos de visión artificial para la detección de objetos a partir de las imágenes obtenidas del escenario.
6. Calcular las coordenadas geográficas de los objetos posteriormente a su identificación.

Los objetos incluidos en el escenario corresponden a balizas terrestres con las características explicadas en el capítulo 3. En concreto, las balizas del concurso incluyen un carácter alfanumérico sobre un fondo con un color determinado. En este proyecto se considerará una primera aproximación en la que se aborda únicamente la identificación de las balizas en el entorno para obtener sus coordenadas, no siendo objeto de estudio la identificación del color o el carácter alfanumérico.

Tras la finalización de este proyecto se dispondrá de una plataforma de simulación completa que permita simular misiones autónomas acorde a las definidas por el concurso *UAS Challenge*.

Esta plataforma de simulación servirá como base de futuros trabajos dentro del proyecto HORUS UPV, para optimizar las estrategias de control y navegación, así como los algoritmos de visión artificial y cálculo de las posiciones de los objetos en un entorno de simulación. Una vez optimizados y validados los algoritmos de control, se dará paso a su implementación en hardware, para ensayos no destructivos, como son las simulaciones *Hardware In The Loop* (HIL), y finalmente su implementación en la aeronave real.

En este proyecto no se pretende por tanto optimizar algoritmos de visión artificial, de control o guiado, sino que se busca la integración del conjunto de algoritmos en un mismo sistema de simulación.

3 | Estudio de necesidades, factores a considerar: limitaciones y condicionantes

El sistema de simulación desarrollado en este trabajo se realiza atendiendo a las características de la misión del concurso *UAS Challenge*. Debido a que no se trabaja con la implementación en hardware de los algoritmos desarrollados, la normativa aplicable es la del concurso. En caso de que se realizara la implementación y verificación de los algoritmos en la plataforma de vuelo real, debe aplicarse, además, la normativa europea de UAS, vigente desde el 31 de diciembre de 2020. En concreto, el Reglamento de Ejecución (UE) 2019/947, referente a las reglas y procedimientos para las operaciones de vehículos aéreos no tripulados, y el Reglamento Delegado (UE) 2019/945, referente a sistemas aéreos no tripulados.

3.1. Normativa del concurso *UAS Challenge*

La normativa aplicable se encuentra en la documentación del concurso, disponible en la página web del iMechE (*Institution of Mechanical Engineers*) [10]. En concreto, son aplicables las especificaciones de la sección 3 de las reglas del concurso (*Design and Operational Requirements*) y el anexo A (*Mission*). El resto de especificaciones corresponden con las fases del diseño, información general sobre el concurso, como la puntuación y las categorías de los premios, al igual que los requisitos técnicos que debe cumplir la aeronave. En esta sección solo se detallan aquellos requisitos que afectan al desarrollo del sistema de simulación de este trabajo.

3.1.1. Requisitos operacionales y de diseño

A continuación se detallan los requisitos operacionales y de diseño que deben tenerse en cuenta en el diseño de la aeronave, lo cual afecta al modelo dinámico utilizado en este trabajo, al igual que en la actuación de la misma.

Requisitos de diseño

1. Configuración de la estructura de la aeronave

La configuración puede ser de ala rotatoria, ala fija u otro tipo, pero el peso máximo al despegue (MTOW - *Maximum Take-Off Weight*) debe ser menor de 10 kg. El tipo de

configuración condiciona la estructura del sistema de control utilizada, en este caso siendo diseñada para una aeronave de ala fija, mientras que la masa de la misma condiciona los parámetros del sistema de control.

2. Sistemas de propulsión

El sistema de propulsión debe ser eléctrico. Este requisito condiciona el modelo de propulsión utilizado en la simulación.

3. Autonomía

La aeronave debe volar de forma completamente autónoma, desde el inicio del despegue hasta el aterrizaje final, incluyendo la navegación por todos los waypoints, el despliegue del paquete de ayuda y la identificación de las balizas terrestres. No obstante, este trabajo se centra solamente en la navegación por los waypoints y la identificación de las balizas, estando las otras fases de vuelo fuera del alcance del mismo.

4. Cámara / Sistema de imagen

La aeronave debería llevar una cámara y tener capacidad de reconocimiento de imágenes para llevar a cabo la búsqueda de las balizas. La posición y el carácter alfanumérico de la baliza debe ser interpretado por la aeronave y enviado a la estación base durante el vuelo. No obstante, en este proyecto solo se analiza la detección y posicionamiento de las balizas.

5. Sistema de navegación

Los organizadores del evento proveen los datos de las coordenadas geográficas de los waypoints de la misión, al igual que las coordenadas que marcan los límites de la zona de vuelo (geocerca). La aeronave debe navegar de forma autónoma alrededor del circuito determinado por los waypoints. El sistema de navegación debe enviar automáticamente una señal para activar el sistema de terminación de vuelo (FTS), en el caso de que la aeronave se desvíe de la zona limitada por la geocerca.

6. Estación de control terrestre

El sistema aéreo no tripulado debe incluir una estación de control terrestre (GCS). Esta debe mostrar y guardar la siguiente información:

- Posición de la aeronave en un mapa.
- Espacio aéreo local, incluyendo la zona delimitada por la geocerca.
- Altura de vuelo (QFE - *Query: Field Elevation*).
- Velocidad indicada (IAS - *Indicated Airspeed*).
- Información sobre el estado de la aeronave.

En el sistema de simulación desarrollado se utilizará la estación de control terrestre únicamente con el fin de definir la misión y visualizar el estado de la aeronave, puesto que toda la información se encuentra disponible como producto de la simulación.

7. Limitación en el uso de productos COTS (*Commercial Off-The-Shelf*)

La estructura de la aeronave y el sistema de control debe ser diseñado desde cero, y no estar basados en sistemas disponibles comercialmente. Los productos COTS permitidos se limitan a motores eléctricos, cámaras, baterías, servos, o plataformas para el desarrollo del control de vuelo, como son *Pixhawk* y *Ardupilot*.

Requisitos operacionales

- La misión se define en el siguiente apartado, y está diseñada para probar distintas capacidades del UAS, necesarias para un sistemas de entrega de paquetes de ayuda humanitaria.
- La aeronave debe ser capaz de despegar y aterrizar en una zona de 30x30 m.
- El tiempo de misión total debe ser menor de 15 minutos
- La aeronave debe ser capaz de operar en vientos de hasta 20 kts, con ráfagas de hasta 25 kts, y lluvia ligera.
- La operación debe llevarse a cabo en condiciones VLOS (*Visual Line of Sight*). Esto es, en un radio horizontal menor a 500 m y por debajo de 400 ft AGL (*Above Ground Level*).

3.1.2. Misión

La misión está compuesta por un desafío central y algunos opcionales que solo pueden completarse una vez ha finalizado el desafío central.

Misión central

El escenario de la misión consiste en la recogida de un paquete de ayuda humanitaria en una base de suministro. La aeronave tiene que transportar esta ayuda a una zona de lanzamiento a través de waypoints definidos, dentro del tiempo establecido. La misión central está compuesta por la fase de preparación previa al vuelo, la entrega del paquete, y el ascenso y descenso.

Desafíos opcionales

Los desafíos opcionales corresponden a una prueba de velocidad, en la cual se mide el tiempo que tarda la aeronave en seguir un circuito limitado por waypoints, sin exceder una velocidad de 60 kts, y la identificación de balizas. Este último se consigue mediante el sistema de procesamiento de imagen, y es lo que se incluye en este trabajo.

4 | Planteamiento de soluciones alternativas y justificación de la solución

4.1. Simulación de sistemas dinámicos

Para la simulación de sistemas dinámicos no se consideran alternativas, puesto que el modelo dinámico utilizado en este trabajo proviene de trabajos previos realizados en el proyecto HORUS-UPV, en el cual se implementó el modelo dinámico en MATLAB[®] y Simulink[®]. Así mismo, si bien podría considerarse el uso de alternativas para simular sistemas dinámicos, como *OpenModelica* o *Scilab*, estos programas no ofrecen las herramientas de las que dispone MATLAB[®] para integrar un entorno de simulación externo o establecer una conexión con estaciones de control terrestres, utilizadas en el sistema de simulación desarrollado en este trabajo. Por lo tanto, la elección de MATLAB[®] y Simulink[®] para el desarrollo del sistema de simulación supone una notoria reducción en el tiempo necesario para llevar a cabo el mismo, con la reducción de costes que ello conlleva, al igual que facilita su desarrollo debido a la amplia documentación disponible.

4.2. Estrategias de control

Existen diversas estrategias de control que pueden aplicarse para desarrollar el sistema de guiado y de control de la aeronave. Algunas de estas son [11]:

1. **Control PID:** es la estrategia de control más utilizada en autopilotos comerciales para plataformas de menor tamaño. La ventaja de esta estrategia se basa en su fácil implementación y ajuste, pudiendo ajustar sus parámetros incluso en vuelo [11].

La acción de control se genera a partir de la diferencia entre un valor medido y un valor de referencia, mediante la suma de 3 acciones, una proporcional (P), una integral (I) y una derivada (D). La acción proporcional depende del valor presente, siendo un amplificador del error instantáneo (cuanto mayor sea el error, mayor será la acción proporcional). La acción integral depende del pasado, pues acumula el error anterior, y busca reducir o eliminar el error estacionario que no puede ser eliminado con la acción proporcional. Por último, la acción derivada busca reducir el error en las variaciones que se puedan producir, como pueden ser perturbaciones externas, proporcional a la velocidad con la que se producen [1]. Con esto, la estructura general de un controlador PID se muestra en la Figura 4.1.

Finalmente, si bien el mecanismo PID es una estrategia de control sencilla y fácil de imple-

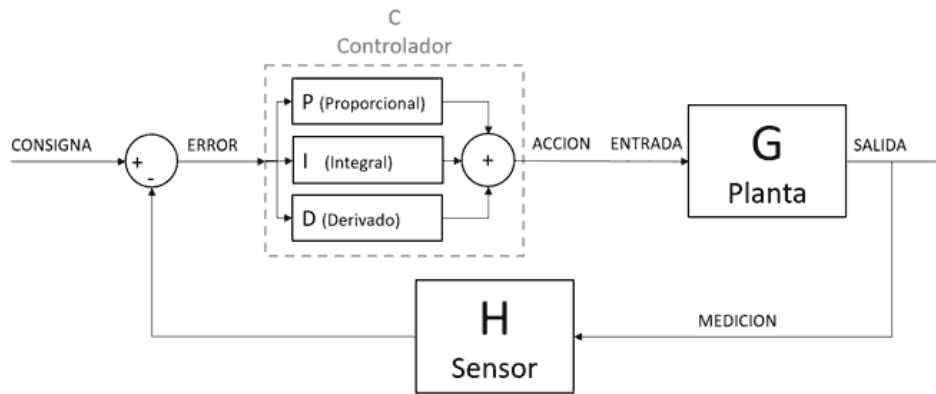


Figura 4.1. Estructura de un controlador PID [1]

mentar, no es tan robusto como otras estrategias de control. Así mismo, según la situación, los parámetros pueden ser difíciles de ajustar.

2. **Fuzzy Based:** como su nombre lo indica, esta estrategia de control está basada en lógica difusa, que, a diferencia de la lógica digital en la cual una variable toma los valores de 0 (*false*) o 1 (*true*), toma valores intermedios entre 0 y 1, representando mejor el mundo real. En este sentido, la lógica difusa se puede considerar una rama de la inteligencia artificial.

A diferencia de los controladores PID, la estrategia de control basada en lógica difusa puede aplicarse a procesos complejos pobremente definidos, bien porque se desconozca su modelo dinámico o bien porque la información proveniente de sus sensores sea imperfecta. En cuanto a la estructura de control, se basa en lazos cerrados al igual que el PID, pero los resultados son más precisos y se obtienen más rápido [2].

La estructura general de un controlador basado en lógica difusa para un sistema SISO (*Single Input-Single Output*) se muestra en la Figura 4.2, donde $r(t)$ es la señal de referencia, $e(t)$ es el error, $u(t)$ es la acción de control e $y(t)$ es la salida del sistema. Los bloques constitutivos del controlador son una Base de Conocimiento (BC), una base de datos (BD), un Motor de Inferencias (MI), un Emborronador (F) y un Desemborronador (D). El Emborronador transforma la entrada determinista en valores subjetivos, que disparan un conjunto de reglas almacenadas en la Base de Conocimiento. El Desemborronador se encarga de combinar el resultado propuesto por cada una de las reglas disparadas, mientras que el Motor de Inferencias concatena las implicancias de la Base de Conocimiento [2].

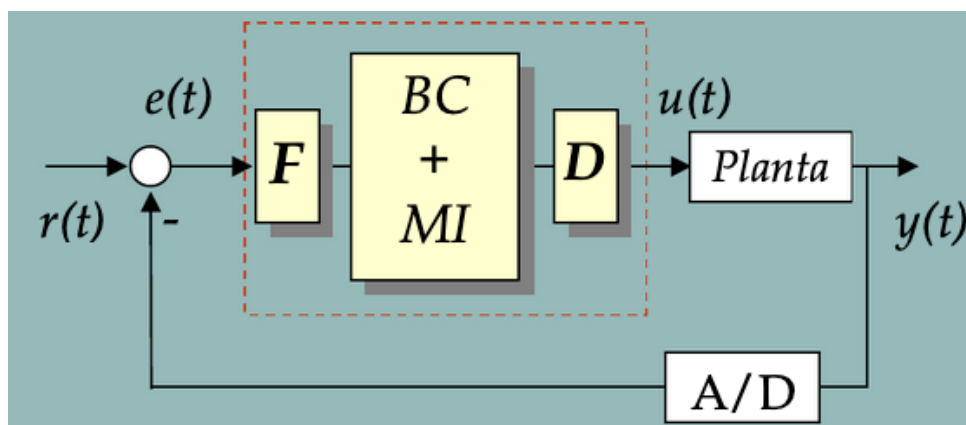


Figura 4.2. Estructura de un controlador basado en lógica difusa [2]

3. **Regulador lineal-cuadrático (LQR)**: está basado en el problema LQ, en el cual la dinámica del sistema está descrita por ecuaciones diferenciales lineales, y el coste, cuyo objetivo es minimizar, está definido por una función cuadrática. Este es uno de los problemas fundamentales en la teoría de control óptimo. Habitualmente, la función de coste está determinada por la suma de las desviaciones de las medidas respecto de los valores deseados [3]. Por consiguiente, el algoritmo encuentra los parámetros necesarios del controlador para minimizar estas desviaciones. En la Figura 4.3 se muestra la estructura del regulador lineal cuadrático, en la cual la ganancia del controlador, $\mathbf{K}(t)$, minimiza la función coste asociada.

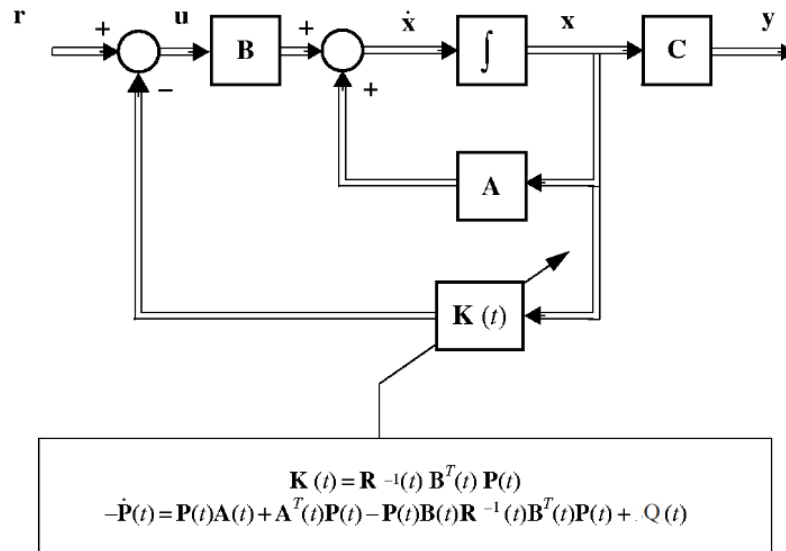


Figura 4.3. Sistema en lazo cerrado con LQR [3]

4. **Control predictivo basado en modelo (MPC)**: es una estrategia de control que utiliza la información sobre el modelo del sistema para predecir las salidas futuras del mismo, utilizándolas como base para optimizar las acciones de control futuras. Esta estrategia de control no es independiente, ya que integra otros métodos de control como el control óptimo y el control de procesos multivariables [4].

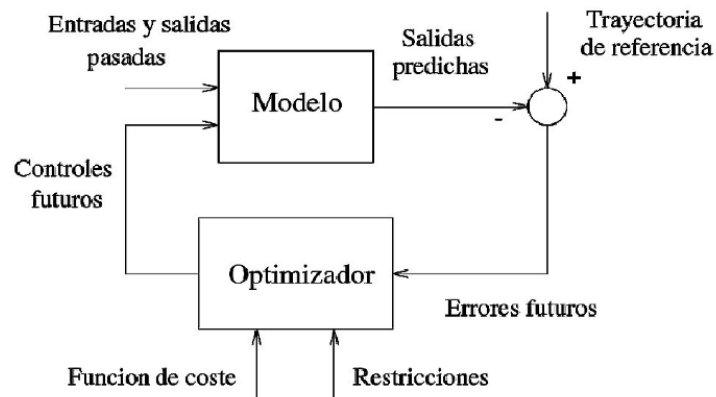


Figura 4.4. Estructura básica del MPC [4]

La estructura necesaria para implementar un control predictivo basado en modelo se da en la Figura 4.4. Las acciones de control futuras son calculadas en el optimizador, a partir de los errores futuros y de la función coste que se quiera minimizar, junto a posibles restricciones sobre las variables del proceso.

5. **Basado en redes neuronales (NN)**: los controladores basados en redes neuronales adaptativas no requieren de un modelo matemático y pueden aplicarse tanto a sistemas SISO (*Single Input-Single Output*) como MIMO (*Multiple Input-Multiple Output*). Las estructura de control se puede diseñar de forma que los pesos de la red neuronal se inicialicen de forma rápida y que esta aprenda en tiempo real.

Un ejemplo del control basado en redes neuronales adaptativas se da en la Figura 4.5. El aprendizaje de la dinámica de la planta se realiza mediante el estimador neuronal, el cual ajusta los pesos de conexión del controlador neuronal que genera la acción de control $u(k)$ [5].

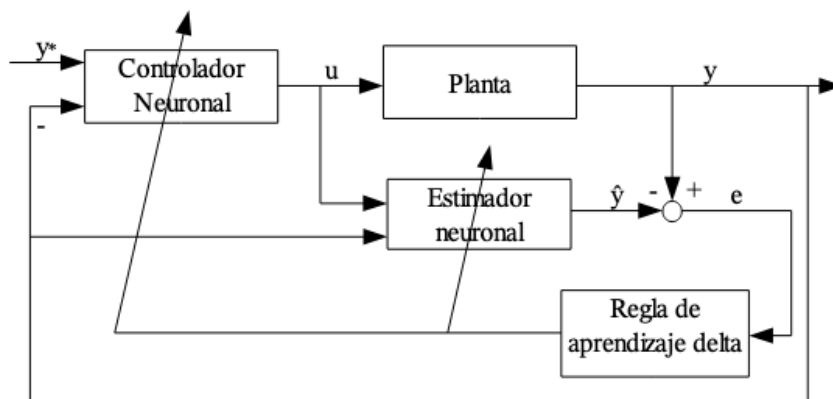


Figura 4.5. Esquema de control neuronal adaptativo [5]

En este trabajo se utilizarán concretamente controladores PID, puesto que tanto el modelo dinámico como el sistema de control utilizado proviene de un trabajo previo realizado dentro del proyecto HORUS-UPV, en el cual se diseñó el sistema de control mediante control PID [12].

4.3. Entorno de simulación

En el entorno de simulación pueden considerarse dos opciones principalmente. La primera consiste en utilizar simuladores de vuelo, como X-Plane o FlightGear, que pueden comunicarse con MATLAB[®] para visualizar el comportamiento de la aeronave. Sin embargo, estos simuladores de vuelo dificultan el uso de las herramientas de simulación para misiones de sistemas aéreos no tripulados proporcionadas por *UAV Toolbox* de MATLAB[®]. En concreto, la obtención de datos del entorno, con especial interés en la obtención de imágenes mediante cámaras incluidas en el entorno de simulación, no es directa, por lo que dificulta la implementación de algoritmos de visión artificial. Debido a esto, se opta por la segunda opción, que consiste en utilizar programas de modelado y motores gráficos para la creación del escenario de simulación. Entre ellos, destacan las siguientes opciones:

- **Unity**¹: es un motor de videojuegos multiplataforma, disponible de forma gratuita mediante el plan personal. Permite crear mundos abiertos interactivos, los cuales podrían ser utilizados para visualizar el comportamiento de la aeronave. Algunas aplicaciones de este motor de videojuegos, además de la creación de juegos, son la creación de películas, animaciones y cinemáticas, o crear aplicaciones en realidad aumentada (AR) y realidad virtual

¹Página web: <https://unity.com/es>

(VR) para visualizar de forma interactiva los productos de una empresa. De igual forma, también tiene su aplicación en el sector aeroespacial, con el fin de reducir el tiempo y costo de desarrollo de tecnologías, o para crear entornos de simulación y entrenamiento.

- **Blender**²: está dedicado al modelado, renderizado, animación y creación de gráficos tridimensionales. Tiene diversas aplicaciones, especialmente en el ámbito arquitectónico e industrial, o por ejemplo la creación de publicaciones tridimensionales en páginas web. Además, es posible establecer una comunicación entre MATLAB[®] y Blender, pudiendo así visualizar el comportamiento de la aeronave durante la misión.
- **Unreal Engine**³: al igual que Unity, es un motor de videojuegos que proporciona una de las herramientas de creación 3D en tiempo real más avanzada. Las aplicaciones de este motor son diversas, desde juegos, cinematografía y simulación hasta arquitectura y transporte.

Si bien los programas explicados anteriormente ofrecen potentes herramientas de diseño 3D para la creación de escenarios y mundos abiertos, en este trabajo se utiliza Unreal Engine, ya que MATLAB[®] ofrece herramientas para la visualizar y verificar algoritmos en escenarios creados con este motor de juegos.

4.4. Estación de control terrestre

La estación de control terrestre es normalmente el software instalado en un ordenador en tierra, que establece la comunicación con el UA, tanto para visualizar en todo momento el estado del mismo como para enviar órdenes de control y modificar la misión definida. Algunas de las estaciones de control terrestre gratuitas disponibles para ordenadores con sistema operativo Windows son *Mission Planner*, *APM Planner 2*, *MAVproxy* y *QGroundControl*, siendo esta última la utilizada. Debido a que todas ofrecen las herramientas necesarias para el desarrollo del sistema de simulación, la decisión se toma por familiaridad con el software, puesto que en el proyecto HORUS UPV se ha trabajado previamente con *QGroundControl*.

4.5. Algoritmos de Visión Artificial

Con el fin de implementar la detección de balizas del concurso al que está enfocado el proyecto HORUS UPV, existen diversas alternativas:

- **OpenCV**: es la librería más conocida, disponible para varias plataformas, y sencilla de utilizar. Cubre estrategias y algoritmos fundamentales para analizar imágenes y vídeos [13].
- **Tensorflow**: es la librería de código abierto para aprendizaje automático (*Machine Learning*) y *Deep Learning* más conocida. Con una API (*Application Programming Interface*) sencilla de utilizar, dispone de modelos preparados para reconocimiento de texto, de objetos, reconocimiento de voz, etcétera [13].

²Página web: <https://www.blender.org/>

³Página web: <https://www.unrealengine.com/en-US/>

- **MATLAB[®]**: ofrece un amplio abanico de funciones y modelo pre-entrenados para aplicaciones como la detección de objetos o el reconocimiento de texto. Es usualmente utilizada en la investigación ya que permite un rápido desarrollo de prototipos [13]. En concreto, esta funcionalidad está disponible a través de *Computer Vision Toolbox*.
- **CUDA** (Compute Unified Device Architecture): es una plataforma de computación en paralelo que permite a los programadores codificar algoritmos en la GPU de NVIDIA. Debido a que una parte significativa de la visión artificial corresponde con el procesamiento de imágenes, aplicación para la cual las GPUs fueron diseñadas, la implementación de algoritmos en la GPU permite un rendimiento mucho mayor [13].
- **YOLO** (You Only Look Once): es un sistema de detección de objetos para procesamiento de imágenes en tiempo real. Diseñada por Joseph Redmon y Ali Farhadi, implementa un algoritmo basado en redes neuronales para procesar toda la imagen, dividiendo la imagen en regiones, prediciendo cuadros de identificación para cada una [13].

La opción finalmente utilizada en este trabajo es la proporcionada por MATLAB[®], ya que la eficiencia del algoritmo no es uno de los objetivos finales de este trabajo, sino la implementación de un algoritmo de visión artificial básico que permita demostrar la capacidad del sistema de simulación para implementar la misión del concurso.

5 | Plataforma de vuelo

5.1. Modelado

El sistema de simulación desarrollado en este trabajo necesita información sobre la posición y orientación de la aeronave, para poder representarla en el entorno de simulación y obtener datos del mismo. Es por esto que se debe disponer de un modelo matemático que represente el comportamiento en vuelo de la aeronave. Además, el modelo matemático permite diseñar y ajustar el sistema de control para misiones autónomas. En este capítulo se explican los resultados de los trabajos previos desarrollados por Daniel Villalibre Vilariño [12] y Álvaro Goterris Fuster [14], que dan lugar al modelo de aeronave utilizado en el presente trabajo.

5.1.1. Sistemas de referencia y actitud de la aeronave

Para obtener las ecuaciones del movimiento de la aeronave, es necesario definir unos sistemas de referencia en los cuales se definen las posiciones, velocidades y aceleraciones de la misma. Consisten generalmente en un triedro ortogonal, y los utilizados en este trabajo son:

- Sistema de referencia en ejes cuerpo ("*Body Axes*"): el triedro representa al sólido rígido de la aeronave. El origen del sistema se sitúa en el centro de masas. Si bien caben distintas direcciones para los ejes, en este trabajo se consideran las siguientes:
 - Eje X_B : eje longitudinal de la aeronave, con sentido positivo hacia el morro de la misma.
 - Eje Y_B : perpendicular al plano de simetría de la aeronave, con sentido positivo hacia la "derecha del piloto".
 - Eje Z_B : perpendicular a los anteriores y sentido positivo hacia abajo, considerando la aeronave en vuelo horizontal.
- Sistema de referencia en ejes viento ("*Wind Axes*"): está ligado a la velocidad aerodinámica instantánea de la aeronave. El eje X_W coincide con el vector de la velocidad aerodinámica. El eje Z_W es perpendicular al eje X_W y se sitúa en el plano de simetría de la aeronave, con sentido "hacia abajo". Por último, el eje Y_W forma un triedro a derechas con los anteriores.
- Sistema de referencia Ejes en Tierra: el origen del mismo se sitúa en una posición fija sobre la superficie terrestre. El eje X_E está dirigido hacia el Norte, el eje Y_E hacia el Este, y el eje Z_E hacia abajo. En la literatura se denomina a este sistema de referencia como NED ("*North-East-Down*"). Una variante del mismo es el ENU ("*East-North-Up*"), donde se mantiene el triedro a derechas, pero con el eje Z apuntando hacia arriba. Este sistema

de referencia será considerado como inercial, respecto al cual se expresará la posición del centro de masas de la aeronave.

- Sistema de referencia Ejes Horizonte Local: tiene su origen en el centro de masas de la aeronave y sus ejes se mantienen paralelos al Sistema Ejes Tierra. Se designan mediante el subíndice H.

Las ecuaciones que describen la dinámica de la aeronave se expresan en ejes estabilidad, que corresponde con una particularización de los ejes cuerpo. Sin embargo, teniendo en cuenta la hipótesis de pequeñas perturbaciones consideradas en el modelo, se considera que el sistema de ejes estabilidad es equivalente al sistema de ejes cuerpo [12].

Actitud de la aeronave

Además de definir la posición de la aeronave en el espacio, dada por las coordenadas espaciales de su Centro de Masas, es necesario definir la orientación de sus ejes cuerpo de referencia, describiendo así su *actitud*.

Existen diversos modos de expresar la orientación de la aeronave (Cosenos Directores, Ángulos de Euler o Cuaterniones). En este trabajo, se utilizan concretamente los ángulos de Euler. Estos ángulos definen la orientación de un sistema de referencia no inercial respecto de uno inercial, mediante tres rotaciones (ángulos) consecutivas alrededor de tres ejes cuerpo. Los ángulos utilizados en mecánica de vuelo son:

- Ángulo de asiento lateral (ϕ): ángulo de rotación sobre el eje X_B .
- Ángulo de asiento longitudinal (θ): ángulo de rotación sobre el eje Y_B .
- Ángulo de rumbo (acimut) (ψ): ángulo de rotación sobre el eje Z_B .

La secuencia de rotaciones utilizada en el ámbito aeronáutico corresponde con la secuencia $\{\psi \rightarrow \theta \rightarrow \phi\}$. A partir de estos ángulos se puede definir la matriz de transformación para pasar de Ejes Tierra a Ejes Cuerpo. De esta forma, la expresión de un vector en Ejes Cuerpo se obtendría premultiplicando el vector definido en Ejes Tierra con la matriz de transformación. Para revertir el cambio se utilizaría la matriz inversa, correspondiente a la traspuesta de la matriz de transformación de Ejes Tierra a Ejes Cuerpo.

5.1.2. Ecuaciones de *Bryan*

Las ecuaciones de Bryan son un conjunto de doce ecuaciones diferenciales de primer orden, no lineales y acopladas, que aproximan el comportamiento de la aeronave con seis grados de libertad, tres de traslación y tres de rotación. Las ecuaciones se obtienen mediante la aplicación de la segunda Ley de Newton, en base a una serie de hipótesis:

1. La aeronave se considera un cuerpo rígido, por lo que la distancia relativa entre dos puntos de la misma es invariante.
2. Las ecuaciones del movimiento de rotación son independientes de las de traslación, pero no al revés.

3. La influencia del giro de la Tierra se considera despreciable.
4. Se considera que la aeronave posee un plano de simetría, siendo los productos de inercia I_{xy} e I_{yx} nulos.
5. No se consideran efectos de inercia rotatoria de la hélice o del motor.
6. La masa de la aeronave permanece constante durante una maniobra.

Estas restricciones no son fundamentales, pudiéndose omitir a costa de un incremento de la complejidad del modelo. Asimismo, en el desarrollo de las ecuaciones se consideraba viento y turbulencias nulas, pero en el modelo desarrollado se incorpora el efecto del viento en la simulación, mediante perturbaciones a la velocidad aerodinámica.

La nomenclatura utilizada en las ecuaciones se da en la Tabla 5.1. Debido a que el desarrollo del modelo dinámico no corresponde con el objetivo de este trabajo, solo se dará la expresión final de estas ecuaciones, estando su desarrollo en la referencia [12].

Eje	Velocidad lineal	Velocidad angular	Fuerza aplicada	Momento aplicado	Distancias	Ángulos
X	u	p	F_x	L	x	ϕ
Y	v	q	F_y	M	y	θ
Z	w	r	F_z	N	z	ψ

Tabla 5.1. Nomenclatura utilizada en las ecuaciones de Bryan.

Ecuaciones de traslación

Se derivan de la segunda Ley de Newton, teniendo en cuenta que la masa de la aeronave es constante ($m\vec{V}_{ABS}$). En este caso, la aceleración absoluta se debe tanto al movimiento de traslación como al de rotación. Las ecuaciones de traslación se dan en las ecuaciones 5.1, 5.2 y 5.3.

$$m \cdot (\dot{u} + q \cdot w - r \cdot v) = F_{TX} + F_{AX} + F_{GX} \quad (5.1)$$

$$m \cdot (\dot{v} + r \cdot u - p \cdot w) = F_{TY} + F_{AY} + F_{GY} \quad (5.2)$$

$$m \cdot (\dot{w} + p \cdot v - q \cdot u) = F_{TZ} + F_{AZ} + F_{GZ} \quad (5.3)$$

Las fuerzas de propulsión que actúan sobre el cuerpo, F_T , se desarrollan en el apartado **Modelo propulsivo**. Las fuerzas aerodinámicas, F_A , se desarrollan en el apartado **Fuerzas y momentos aerodinámicos**. En cuanto a las fuerzas gravitatorias en Ejes Cuerpo, F_G , estas se obtienen aplicando la matriz de rotación que transforma un vector en Ejes Horizonte Local a Ejes Cuerpo, resultando en:

$$\vec{F}_G = m \cdot g \cdot \begin{pmatrix} -\sin \theta \\ \cos \theta \sin \phi \\ \cos \theta \cos \phi \end{pmatrix} \quad (5.4)$$

Ecuaciones de rotación

Las ecuaciones de rotación relacionan los momentos aerodinámicos (M_A) y propulsivos (M_T) con las aceleraciones angulares. Se obtienen aplicando el principio de Dinámica de Cuerpos Rígidos, que establece que la derivada temporal absoluta del momento angular es igual al momento total externo aplicado sobre la aeronave ($\vec{M} = \{L, M, N\} = \dot{\vec{H}}_{ABS}$). En forma matricial, las ecuaciones vienen dadas por:

$$\begin{bmatrix} L \\ M \\ N \end{bmatrix} = \begin{bmatrix} I_{xx} & 0 & -I_{xz} \\ 0 & I_{yy} & 0 \\ -I_{xz} & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} + \begin{bmatrix} 0 & -r & 0 \\ r & 0 & -p \\ 0 & p & 0 \end{bmatrix} \begin{bmatrix} I_{xx} & 0 & -I_{xz} \\ 0 & I_{yy} & 0 \\ -I_{xz} & 0 & I_{zz} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \quad (5.5)$$

Estas ecuaciones se suelen escribir de forma que en cada una aparezca a la izquierda solamente la derivada temporal de una velocidad angular. Esto es,

$$\dot{p} = \frac{I_{zz}}{A} L + \frac{I_{xz}}{A} N + \left(\frac{I_{xz}(I_{xx} - I_{yy} + I_{zz})}{A} \right) p \cdot q + \left(\frac{I_{zz}(I_{yy} - I_{zz}) - I_{xz}^2}{A} \right) r \cdot q \quad (5.6)$$

$$\dot{q} = \frac{M}{I_{yy}} + \frac{I_{zz} - I_{xx}}{I_{yy}} p \cdot r + \frac{I_{xz}}{I_{yy}} (r^2 - p^2) \quad (5.7)$$

$$\dot{r} = \frac{I_{xx}}{A} N + \frac{I_{xz}}{A} L + \left(\frac{I_{xx}(I_{xx} - I_{yy}) + I_{xz}^2}{A} \right) p \cdot q + \left(\frac{I_{xz}(I_{yy} - I_{xx} - I_{zz})}{A} \right) r \cdot q \quad (5.8)$$

Donde,

$$\begin{aligned} A &= I_{xx} I_{zz} - I_{xz}^2 \\ L &= M_{AX} + M_{TX} \\ M &= M_{AY} + M_{TY} \\ N &= M_{AZ} + M_{TZ} \end{aligned}$$

Relación entre la velocidad angular y las derivadas de los ángulos de Euler

Este conjunto de ecuaciones relaciona los ángulos de Euler y la velocidad angular en Ejes Cuerpo. Es de utilidad puesto que, mediante giróscopos montados sobre los ejes X_B , Y_B , Z_B de la aeronave, se pueden medir los ángulos de Euler. A partir de estos datos sería inmediata la obtención de las velocidades angulares en Ejes Cuerpo, necesarias para resolver las ecuaciones de traslación y rotación. Las relaciones vienen dadas por las ecuaciones 5.9, 5.10 y 5.11.

$$p = \dot{\phi} - \dot{\psi} \sin \theta \quad (5.9)$$

$$q = \dot{\theta} \cos \phi + \dot{\psi} \cos \theta \sin \phi \quad (5.10)$$

$$r = \dot{\psi} \cos \theta \cos \phi - \dot{\theta} \sin \phi \quad (5.11)$$

Ecuaciones cinemáticas

Estas ecuaciones completan el desarrollo de las ecuaciones necesarias para el estudio del comportamiento de la aeronave. Se obtienen a partir de la rotación de las velocidades en Ejes

Cuerpo, resultando en las siguientes expresiones:

$$\dot{x}_E = u \cos \psi \cos \theta + v(\cos \psi \sin \theta \sin \phi - \cos \phi \sin \psi) + w(\sin \theta \cos \phi \cos \psi + \sin \phi \sin \psi) \quad (5.12)$$

$$\dot{y}_E = u \cos \theta \sin \psi + v(\cos \phi \cos \psi + \sin \theta \sin \phi \sin \psi) + w(-\cos \psi \sin \phi + \cos \phi \sin \theta \sin \psi) \quad (5.13)$$

$$\dot{z}_E = -u \sin \theta + v(\cos \theta \sin \phi) + w(\cos \theta \cos \phi) \quad (5.14)$$

5.1.3. Características de la plataforma

En este apartado se resumen las principales características de la aeronave utilizada para la simulación. Todos los datos se han extraído del trabajo previo realizado por Álvaro Goterris Fuster [14], dentro del proyecto HORUS-UPV.

Características geométricas

En este trabajo solo son de interés las características geométricas dadas en la Tabla 5.2, puesto que son las necesarias para calcular las fuerzas y momentos aerodinámicos dados en el apartado Fuerzas y momentos aerodinámicos.

b_w	c_w	S_w	d_{Th}
2.00 m	0.25 m	0.50 m ²	48 mm

Tabla 5.2. Propiedades geométricas de la aeronave.

Propiedades másicas e inerciales

Las propiedades másicas e inerciales de la aeronave son necesarias para el cómputo de las fuerzas y los momentos que actúan sobre la misma. Los valores considerados en este trabajo para la simulación de la dinámica de la aeronave se dan en la Tabla 5.3. En cuanto a los productos de inercia, solo se considera I_{xx} , debido al plano de simetría considerado, según lo explicado en la hipótesis 4 del apartado **Ecuaciones de Bryan**.

m	I_{xx}	I_{yy}	I_{zz}	I_{xz}
7.4430 kg	0.6087 kg m ²	1.2940 kg m ²	1.7180 kg m ²	0.093 32 kg m ²

Tabla 5.3. Propiedades másicas e inerciales de la aeronave.

Derivadas aerodinámicas

Los valores de las derivadas aerodinámicas considerados en este trabajo son los dados en las Tablas 5.4 y 5.5. Estos valores se pueden obtener mediante métodos teórico-empíricos, CFD (Mecánica de Fluidos Computacional), o mediante ensayos de vuelo. Los valores considerados en este trabajo fueron obtenidos mediante el programa XFLR5, desarrollado y actualizado de acuerdo a las normas GPL (*General Public License*).

variable	α	β	$\dot{\alpha}$	$\dot{\beta}$	V	p	q	r
CL	4.8406	0	2.2396	0	0	0	10.1570	0
CX	(*)	0	–	0	0	0	–	0
CY	0	-0.1437	0	0	0	0.0398	0	0.1738
CZ	–	0	–	0	0	0	–	0
Cl	0	-0.0207	0	0	0	-0.5269	0	0.2224
CM	-1.7800	0	-9.4711	0	0	0	-24.8790	0
CN	0	0.0756	0	0	0	-0.1466	0	-0.0894

Tabla 5.4. Derivadas aerodinámicas de las variables de estado

variable	δA	δE	δF	δR
CX	0	-0.0217	-0.0902	0
CY	-0.0155	0	0	0.1201
CZ	0	-0.5551	-1.6182	0
Cl	0.4548	0	0	-0.0024
CM	0	-2.2135	0.7259	0
CN	0.0082	0	0	-0.0673

Tabla 5.5. Derivadas aerodinámicas de las variables de control

Los coeficientes restantes, correspondientes a la polar (coeficientes de resistencia) y coeficientes parásitos, no se han incluido en las tablas anteriores por ser independientes de las variables de estado o depender de una sola variable. Sus valores son:

$$CD_0 = 0.030 \quad CD_1 = 0.0907 \quad CD_2 = 1.4201 \quad CL_0 = 0.3310 \quad CM_0 = 0.2662$$

5.1.4. Modelo propulsivo

El modelo propulsivo utilizado en este trabajo se obtuvo a partir de datos experimentales proporcionados por el fabricante del motor eléctrico utilizado. El proceso de obtención del mismo se detalla en el trabajo realizado por Daniel Villalibre Vilariño [12], dando como resultado:

$$Th(\delta_P) = 55.784 \cdot \delta_P^2 + 10.972 \cdot \delta_P \quad (5.15)$$

La fuerza del empuje actúa únicamente sobre el eje X_B , dada la geometría de la aeronave y la disposición de la planta propulsiva en la misma. En cuanto al momento producido por el empuje, ya que el punto de aplicación se encuentra por encima del Centro de Masas, producirá un momento en el eje Y_B , proporcional a la distancia d_{Th} a la que se encuentre el punto de aplicación. En este caso, $d_{Th} = 48$ mm, y las ecuaciones de fuerzas y momentos propulsivos

vienen dadas por:

$$\vec{F}_T = \begin{bmatrix} F_{TX} \\ F_{TY} \\ F_{TZ} \end{bmatrix} = \begin{bmatrix} Th \\ 0 \\ 0 \end{bmatrix} \quad \vec{M}_T = \begin{bmatrix} M_{TX} \\ M_{TY} \\ M_{TZ} \end{bmatrix} = \begin{bmatrix} 0 \\ -d_{Th} \cdot Th \\ 0 \end{bmatrix} \quad (5.16)$$

5.1.5. Fuerzas y momentos aerodinámicos

En el estudio aerodinámico se utilizan una serie de coeficientes adimensionales, conocidos como coeficientes aerodinámicos, para calcular las fuerzas y momentos que sufre un cuerpo en movimiento en el aire. Debido a que se asume la hipótesis de pequeñas perturbaciones, el modelo aerodinámico se puede simplificar a un sistema lineal. De esta forma, los coeficientes de fuerza y sustentación vienen dados por la suma de una serie de coeficientes, denominados derivadas aerodinámicas, multiplicados por la variable de derivación. Con esto, las ecuaciones de fueras y momentos en ejes cuerpo son:

$$F_{AX} = \frac{1}{2} \rho S_w V^2 C_X \quad (5.17)$$

$$F_{AY} = \frac{1}{2} \rho S_w V^2 C_Y \quad (5.18)$$

$$F_{AZ} = \frac{1}{2} \rho S_w V^2 C_Z \quad (5.19)$$

$$M_{AX} = \frac{1}{2} \rho S_w b_w V^2 C_l \quad (5.20)$$

$$M_{AY} = \frac{1}{2} \rho S_w c_w V^2 C_M \quad (5.21)$$

$$M_{AZ} = \frac{1}{2} \rho S_w b_w V^2 C_N \quad (5.22)$$

$$(5.23)$$

Las fuerzas aerodinámicas se calculan habitualmente en ejes viento, a partir de los coeficientes de sustentación y resistencia dados en las ecuaciones 5.24 y 5.25. Sin embargo, en las ecuaciones de Bryan descritas anteriormente, es necesario expresar las fuerzas aerodinámicas en ejes cuerpo. Para ello se utilizan los coeficientes aerodinámicos de las ecuaciones 5.26, 5.27 y 5.28. Estos dependen de los coeficientes de sustentación y resistencia, de las variables de estado y las variables de control.

$$C_L = C_{L_0} + C_{L_\alpha} \cdot \alpha + C_{L_{\delta E}} \cdot \delta_E + \frac{c_w}{2 \cdot V} \cdot (C_{L_{aDot}} \cdot \dot{\alpha} + C_{L_q} \cdot q) \quad (5.24)$$

$$C_D = C_{D_0} + C_{D_1} \cdot \alpha + C_{D_2} \cdot \alpha^2 \quad (5.25)$$

$$C_X = -\frac{\cos \alpha}{\cos \beta} \cdot C_D - \cos \alpha \cdot \tan \beta \cdot C_Y + \sin \alpha \cdot C_L \quad (5.26)$$

$$C_Y = C_{Y_\beta} \cdot \beta + C_{Y_{\delta A}} \cdot \delta_A + C_{Y_{\delta R}} \cdot \delta_R + \frac{b_w}{2 \cdot V} \left(C_{Y_{\beta Dot}} \cdot \dot{\beta} + C_{Y_p} \cdot p + C_{Y_r} \cdot r \right) \quad (5.27)$$

$$C_Z = -\frac{\sin \alpha}{\cos \beta} \cdot C_D - \sin \alpha \cdot \tan \beta \cdot C_Y - \cos \alpha \cdot C_L \quad (5.28)$$

Por otro lado, los momentos aerodinámicos se calculan a partir de los coeficientes dados por las ecuaciones 5.29, 5.30 y 5.31, donde el coeficiente de momento de alabeo se expresa con ele

minúscula para evitar confusión con el coeficiente de sustentación (C_L).

$$C_l = C_{l_\beta} \cdot \beta + C_{l_{\delta A}} \cdot \delta_A + C_{l_{\delta R}} \cdot \delta_R + \frac{b_w}{2 \cdot V} \cdot \left(C_{l_{\beta \text{Dot}}} \cdot \dot{\beta} + C_{l_p} \cdot p + C_{l_r} \cdot r \right) \quad (5.29)$$

$$C_M = C_{M_0} + C_{M_\alpha} \cdot \alpha + C_{M_{\delta E}} \cdot \delta_E + \frac{c_w}{2 \cdot V} \cdot \left(C_{M_{\alpha \text{Dot}}} \cdot \dot{\alpha} + C_{M_q} \cdot q \right) \quad (5.30)$$

$$C_N = C_{N_\beta} \cdot \beta + C_{N_{\delta A}} \cdot \delta_A + C_{N_{\delta R}} \cdot \delta_R + \frac{b_w}{2 \cdot V} \cdot \left(C_{N_{\beta \text{Dot}}} \cdot \dot{\beta} + C_{N_p} \cdot p + C_{N_r} \cdot r \right) \quad (5.31)$$

Los coeficientes aerodinámicos de las ecuaciones anteriores dependen de la velocidad aerodinámica (ecuación 5.32), el ángulo de ataque (ecuación 5.33) y el ángulo de derrape (ecuación 5.34), entre otros. Estas variables dependen de la velocidad relativa entre la aeronave y la masa de aire. Si se considera una atmósfera en calma, la velocidad relativa será igual a la velocidad de la aeronave, pero en el modelo considerado se incluyen perturbaciones atmosféricas para simular el viento. Por tanto, la velocidad relativa será la diferencia entre la velocidad de la aeronave y la velocidad del viento. Esto es, si la aeronave se encuentra con viento en cara, la velocidad relativa será mayor que la velocidad de la aeronave, y menor si se encuentra con viento en cola.

$$V = \sqrt{u_r^2 + v_r^2 + w_r^2} \quad (5.32)$$

$$\alpha = \arctan \frac{w_r}{u_r} \quad (5.33)$$

$$\beta = \arcsin \frac{v_r}{V} \quad (5.34)$$

Los valores de las derivadas aerodinámicas utilizadas para el cálculo de los coeficientes de fuerzas y momentos aerodinámicos están en el apartado de **Características de la plataforma**.

5.2. Sistema de control

Las normas del concurso, como se ha explicado en el estudio de necesidades, establecen que el UAS debe ser capaz de realizar el seguimiento de los waypoints que definen la misión de forma autónoma. Para ello, la etapa que sigue al modelado dinámico de la aeronave es el diseño del sistema de control. En este trabajo, el sistema de control utilizado es el desarrollado por Daniel Villalibre Vilariño [12]. En esta sección se explicará el proceso de diseño de este sistema de control, de modo que pueda comprenderse la implementación explicada en el capítulo 6.

5.2.1. Arquitectura

La arquitectura del sistema de control empleado corresponde con un modelo en cascada, compuesto por dos lazos de control, uno externo y otro interno, como se muestra en la Figura 5.1. El lazo de control externo corresponde con el guiado de la aeronave, mientras que el lazo de control interno corresponde al control de la misma mediante las superficies de control.

El lazo de control externo genera la referencia del ángulo de cabeceo necesario para conseguir la altura especificada en la misión, la referencia del ángulo de alabeo para conseguir el rumbo especificado, la velocidad de referencia (u) de acuerdo a la posición de la aeronave respecto a la misión, y la referencia de la aceleración lateral (\dot{v}). De las cuatro referencias generadas, solo las del ángulo de cabeceo y alabeo requieren de controladores PID. La aceleración lateral de referencia será siempre cero y la velocidad de referencia dependerá de la velocidad especificada

en el waypoint de destino y el waypoint previo. Cabe destacar que las referencias utilizadas por el lazo de control externo (rumbo, altura, velocidad) se obtienen a partir de la ley de navegación utilizada, la cual se explica en el capítulo 6.

Por el contrario, el lazo de control interno dispone de cuatro controladores PID, que generan las actuaciones en los alerones, los elevadores, la palanca de gases y el timón de cola, de acuerdo a las diferencias entre las referencias de los ángulos de cabeceo y alabeo, la velocidad y la aceleración lateral, respecto de los valores reales en un instante dado.

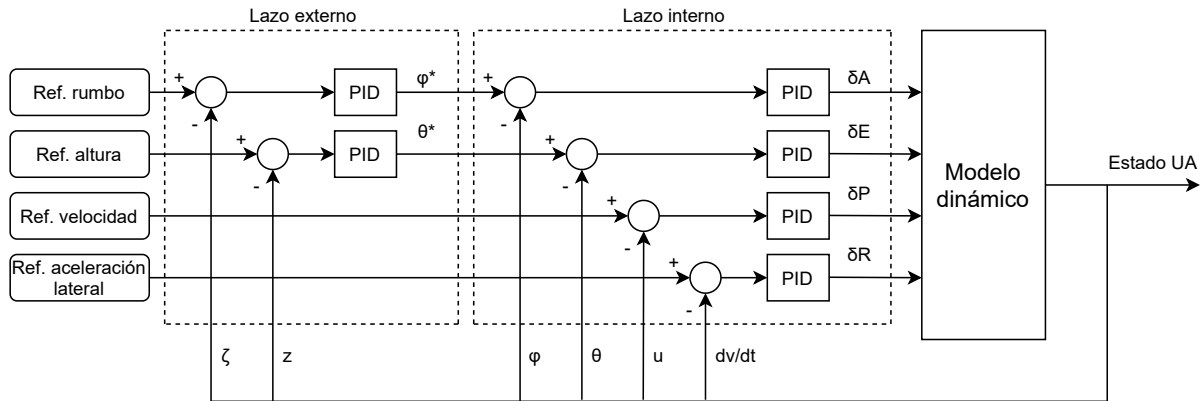


Figura 5.1. Arquitectura del sistema de control

5.2.2. Linealización del modelo

El primer paso para el diseño del sistema de control es la linealización del modelo dinámico. Si bien este modelo permite aproximar el comportamiento de la aeronave en condiciones de vuelo real, no es posible aplicar estrategias de análisis dada su naturaleza no lineal y el acoplamiento entre ecuaciones. Por este motivo, es necesario linealizar el sistema y obtener un modelo lineal que se aproxime al modelo no lineal alrededor de un punto de funcionamiento. El proceso seguido es:

1. Seleccionar el punto de funcionamiento.
2. Obtener las matrices de espacio de estados del modelo.
3. Obtener las funciones de transferencia entre las variables de control y las variables de estado.

Cabe destacar que en este trabajo no se incluirá la comparativa del modelo linealizado con el modelo no lineal. Este es un paso fundamental para cerciorarse de que el modelo linealizado es equivalente al modelo no lineal en el entorno del punto de funcionamiento. Sin embargo, este proceso no aporta información relevante para este trabajo, pudiéndose consultar en la referencia [12].

Selección del punto de funcionamiento

La obtención del modelo linealizado se basa en el desarrollo en serie de Taylor de las variables del sistema. Para ello, es necesario un punto de funcionamiento que se encuentre dentro del

dominio de vuelo y que sea lo más próximo a las condiciones en las que se desarrollará la misión del concurso. En este punto de funcionamiento existe un equilibrio de fuerzas y momentos, por lo que la aeronave es capaz de mantener un vuelo nivelado sin ninguna acción en las superficies de control, en ausencia de perturbaciones externas.

Cabe destacar que el modelo linealizado solo tendrá validez en condiciones cercanas a las del punto de funcionamiento, disminuyendo su precisión cuanto mayor sea la diferencia de las condiciones del vuelo respecto a las del punto de funcionamiento. Además, el modelo linealizado trabaja con incrementos respecto a los valores de las variables de estado en el punto de funcionamiento, y no con valores absolutos. Por lo tanto, las acciones determinadas por los controladores corresponderán con incrementos respecto al equilibrio, siendo necesario sumar los incrementos a los valores del punto de funcionamiento antes de accionar las superficies de control.

Los valores en el punto de funcionamiento de las variables de estado consideradas se dan en la Tabla 5.6.

u	24.9999 m s^{-1}	p	0 rad s^{-1}	ϕ	0 rad	x	0 m
v	0 m s^{-1}	q	0 rad s^{-1}	θ	-0.0020 rad	y	0 m
w	-0.0500 m s^{-1}	r	0 rad s^{-1}	ψ	0 rad	z	-150 m

Tabla 5.6. Valor de las variables de estado en el punto de funcionamiento

Obtención de las matrices de espacio de estados

La representación en espacio de estados expresa la relación entre las variables de entrada (\vec{u}), las variables de estado (\vec{x}) y las salidas del sistema (\vec{y}). El conjunto de ecuaciones se expresa en forma matricial de acuerdo a la forma:

$$\begin{aligned}\dot{\vec{x}} &= A \cdot \vec{x} + B \cdot \vec{u} \\ \vec{y} &= C \cdot \vec{x} + D \cdot \vec{u}\end{aligned}\tag{5.35}$$

Las entradas del sistema corresponden con la deflexión de las superficies de control, por lo que $\vec{u} = \{\delta_A, \delta_E, \delta_P, \delta_R\}^T$, donde se tiene en cuenta la deflexión de los alerones, de los elevadores, de la palanca de gases y del timón de cola, respectivamente. Las variables de estado son las tres velocidades lineales, las tres velocidades angulares y los ángulos de Euler, $\vec{x} = \{u, v, w, p, q, r, \phi, \theta, \psi\}^T$. Por último, las salidas que definen al sistema son las variables de estado, más las tres velocidades lineales en ejes tierra y el rumbo de la aeronave, $\vec{y} = \{u, v, w, p, q, r, \phi, \theta, \psi, \dot{x}, \dot{y}, \dot{z}, \zeta\}^T$, donde el rumbo viene dado por la ecuación 5.36.

$$\zeta = \arctan \frac{\dot{y}}{\dot{x}}\tag{5.36}$$

Los valores de las matrices A y B de espacio de estados se obtienen mediante MATLAB[®]. Para ello, se define un sistema de matrices unilateral, con todos los términos de la ecuación en un solo miembro:

$$MI \cdot \dot{\vec{x}} - MD \cdot \vec{x} - MC \cdot \vec{u} = 0\tag{5.37}$$

Donde las matrices se obtienen por desarrollo de Taylor sobre las ecuaciones dinámicas explicadas en el apartado de **Ecuaciones de Bryan**, sin considerar las ecuaciones cinemáticas:

$$MI(i, j) = \frac{\partial \text{ecuacion}_i}{\partial \dot{x}_j} \quad (5.38)$$

$$MD(i, j) = \frac{\partial \text{ecuacion}_i}{\partial x_j} \quad (5.39)$$

$$MC(i, j) = \frac{\partial \text{ecuacion}_i}{\partial u_j} \quad (5.40)$$

Con esto, las matrices A y B vienen dadas por la expresión 5.41.

$$\begin{aligned} A &= -MI^{-1} \cdot MD \\ B &= -MI^{-1} \cdot MC \end{aligned} \quad (5.41)$$

La matriz D es una matriz de ceros de dimensiones 13x4, ya que las variables de salida se obtienen a partir de las variables de estado. Por otro lado, la matriz C se obtiene aplicando un proceso similar al anterior. Las primeras 9 filas y las 9 columnas de la matriz C corresponden a la matriz identidad de dimensiones 9x9, ya que las salidas son las mismas variables de estado. Por otro lado, las últimas 4 columnas se obtienen aplicando un proceso análogo al explicado para las matrices A y B, pero teniendo en cuenta las ecuaciones cinemáticas y la definición del rumbo.

Este proceso da como resultado la representación del sistema dinámico mediante espacio estados, con matrices constantes. A partir de estas se obtienen las funciones de transferencia necesarias para ajustar los controladores.

Obtención de las funciones de transferencia

Las funciones de transferencia son un modelo matemático que relacionan la respuesta del sistema o una señal, respecto a una variable de entrada. Esto permite obtener información acerca de la respuesta según la entrada aplicada, a partir de lo cual se ajustan los controladores para que la respuesta sea la deseada. Debido a que el sistema de control emplea 6 controladores PID, como se ha explicado en el apartado 5.2.1, se obtienen las 6 funciones de transferencia explicadas a continuación, mediante la función `ss2tf` de MATLAB[®].

Función de transferencia ϕ/δ_A

La respuesta del ángulo de alabeo respecto de la deflexión de los alerones viene dada por la función de transferencia con la siguiente expresión:

$$\frac{\phi(s)}{\delta_A(s)} = \frac{284.5 \cdot s^2 + 275.6 \cdot s + 4743}{s^4 + 14.01 \cdot s^3 + 35.45 \cdot s^2 + 240.4 \cdot s - 32.06} \quad (5.42)$$

Los polos y ceros de esta función de transferencia pueden observarse en la Tabla 5.7. El par de polos conjugados de esta función de transferencia indica un carácter oscilatorio de la respuesta del ángulo de alabeo. Por otro lado, existe un polo real positivo, lo que indica inestabilidad en el sistema. Sin embargo, esta inestabilidad podrá ser corregida por la acción de control.

	Parte real	Parte imaginaria
Polos	0.1307	0
	-0.7082	-4.3333
	-0.7082	4.3333
	-12.7235	0
Ceros	-0.4843	-4.0540
	-0.4843	4.0540

Tabla 5.7. Polos y ceros de la función de transferencia ϕ/δ_A

Función de transferencia θ/δ_E

La relación entre el ángulo de cabeceo y la deflexión de los elevadores viene descrito por la siguiente función de transferencia:

$$\frac{\theta(s)}{\delta_E(s)} = \frac{-79.72 \cdot s^2 - 364.8 \cdot s - 47.5}{s^4 + 11.12 \cdot s^3 + 84.14 \cdot s^2 + 8.197 \cdot s + 20.8} \quad (5.43)$$

De forma similar al caso anterior, esta función de transferencia presenta cuatro polos y dos ceros, dados en la Tabla 5.8. Sin embargo, la parte real de todos sus polos es negativa, por lo que presenta una dinámica estable con oscilaciones en el periodo transitorio, debido a los dos pares de polos conjugados.

	Parte real	Parte imaginaria
Polos	-0.0327	-0.4991
	-0.0327	0.4991
	-5.5291	-7.2521
	-5.291	7.2521
Ceros	-0.1341	0
	-4.4424	0

Tabla 5.8. Polos y ceros de la función de transferencia θ/δ_E

Función de transferencia u/δ_P

La respuesta de la velocidad frente a un cambio en la posición de la palanca de gases viene determinada por la siguiente función de transferencia:

$$\frac{u(s)}{\delta_P(s)} = \frac{5.628 \cdot s + 0.9127}{s^2 + 0.06531 \cdot s + 0.2501} \quad (5.44)$$

En este caso, la función de transferencia se da de forma simplificada, donde se cancelan los polos y ceros próximos entre sí. Los polos y ceros resultantes son los dados en la Tabla 5.9.

	Parte real	Parte imaginaria
Polos	-0.0327	-0.4991
	-0.0327	0.4991
Ceros	-0.1622	0

Tabla 5.9. Polos y ceros de la función de transferencia u/δ_P Función de transferencia \dot{v}/δ_R

La relación entre la aceleración lateral y la deflexión del timón de cola viene dada por la siguiente función de transferencia:

$$\frac{\dot{v}(s)}{\delta_R(s)} = \frac{3.044 \cdot s^2 + 374.7 \cdot s - 62.7}{s^2 + 1.416 \cdot s + 19.28} \quad (5.45)$$

Esta función de transferencia presentaba originalmente los mismos polos que la función de transferencia dada por la ecuación 5.42. Sin embargo, existían dos ceros con valor similar a los polos reales, pudiéndose simplificar la expresión. De esta forma, los polos y ceros de esta función de transferencia son los dados en la Tabla 5.10.

	Parte real	Parte imaginaria
Polos	-0.7082	-4.333
	-0.7082	4.333
Ceros	0.1671	0
	-123.2638	0

Tabla 5.10. Polos y ceros de la función de transferencia \dot{v}/δ_R Función de transferencia z/θ

Mediante un cambio en el ángulo de cabeceo, es posible incrementar o disminuir la altura de la aeronave, por lo que la obtención de la función de transferencia que relaciona la altura con el ángulo de cabeceo es fundamental para diseñar el controlador. Esta función de transferencia viene dada por:

$$\frac{z(s)}{\theta(s)} = \frac{-112.4 \cdot s - 7.153}{s \cdot (s^2 + 4.576 \cdot s + 0.5957)} \quad (5.46)$$

Donde se ha simplificado eliminando un cero de fase no mínima y un cero situado lejos del origen, ya que no son relevantes para calcular el controlador. Por tanto, los polos y ceros de esta función de transferencia vienen dados por la Tabla 5.11.

Función de transferencia ζ/ϕ

En las aeronaves, el cambio de rumbo se produce mediante un cambio en el ángulo de alabeo, conseguido mediante la deflexión de los alerones y el timón de cola para mantener un giro coordinado. En consecuencia, es necesario obtener la función de transferencia que relacione un

	Parte real	Parte imaginaria
Polos	0	0
	-0.1341	0
	-4.4424	0
Ceros	-0.0636	0

Tabla 5.11. Polos y ceros de la función de transferencia z/θ

cambio de rumbo según un cambio en el ángulo de alabeo. De esta forma, se podrá diseñar un controlador para que la aeronave sea capaz de seguir los waypoints, modificando su rumbo acorde a la misión. Esta función de transferencia viene dada por:

$$\frac{\zeta(s)}{\phi(s)} = \frac{0.061 \cdot s^3 - 1.195 \cdot s^2 - 0.195 \cdot s + 6.542}{s \cdot (s^2 + 0.969 \cdot s + 16.67)} \quad (5.47)$$

Los polos y ceros de esta función de transferencia son los dados en la Tabla 5.12. Se puede observar que la función de transferencia presenta dos ceros de fase no mínima, asociados a dinámicas rápidas y lentas de acuerdo a su magnitud.

	Parte real	Parte imaginaria
Polos	0	0
	-0.4843	-4.0540
	-0.4843	4.0540
Ceros	19.5924	0
	2.4055	0
	-2.2885	0

Tabla 5.12. Polos y ceros de la función de transferencia ζ/ϕ

5.2.3. Ajuste de los controladores

Los controladores utilizados para el sistema de control, como ya se ha mencionado, corresponden con controladores de tipo PID, cuya representación se da en la ecuación 5.48. El ajuste de los mismos se realiza mediante la aplicación *pidTuner*, proporcionada por MATLAB[®], mediante la cual se obtienen las ganancias de los controladores atendiendo a las especificaciones que se requieran de los mismos.

$$PID(s) = K_p + K_i \cdot \frac{1}{s} + K_d \cdot \frac{1}{1 + \frac{1}{s}} \quad (5.48)$$

Cabe destacar que las características de la respuesta real del sistema, como el tiempo de establecimiento o la sobreoscilación, pueden diferir respecto a las obtenidas con *pidTuner*, debido a que se utiliza una aproximación linealizada del modelo dinámico.

Todos los controladores se encuentran limitados a un valor máximo atendiendo a las características físicas de la aeronave, concretamente a las características estructurales y aerodinámicas. Los valores máximos son los dados en la Tabla 5.13.

Variable	Valor mínimo	Valor máximo
Alerones	-30°	+30°
Elevadores	-30°	+30°
Timón de cola	-30°	+30°
Ángulo de alabeo	-60°	+60°
Ángulo de cabeceo	-15°	+15°
Palanca de gases	0	1

Tabla 5.13. Limitaciones de las acciones de control

Lazo de control interno

El lazo de control interno dispone de cuatro controladores PID, de acuerdo a la estructura dada en la Figura 5.1, cuyas ganancias se dan en la Tabla 5.14. Todos los controladores generarán los incrementos que deben aplicarse a las superficies de control o a la palanca de gases para reducir el error a cero, es decir, que el valor real de las variables y las referencias sean iguales. Los controladores del ángulo de cabeceo y alabeo tienen una estructura PID, con ganancia proporcional, integral y derivativa. Por el contrario, los controladores de la velocidad y aceleración lateral tienen una estructura PI, únicamente con ganancia proporcional e integral.

	K_p	K_i	K_d
Ángulo de cabeceo	-12.823	-21.480	-1.913
Ángulo de alabeo	3.193	10.330	0.247
Aceleración lateral	0.149	43.638	0
Velocidad	1.370	0.186	0

Tabla 5.14. Ganancias de los controladores del lazo interno

Lazo de control externo

En el lazo de control externo solo se utilizan dos controladores PID, como puede observarse en la Figura 5.1, ya que la referencia del control de la aceleración lateral es nula, mientras que la velocidad de referencia se obtiene directamente de la posición de la aeronave respecto a la trayectoria definida en la misión.

Las ganancias de los controladores utilizados para obtener las referencias del ángulo de alabeo y cabeceo son las dadas en la Tabla 5.15. El controlador del rumbo permite obtener la referencia del ángulo de alabeo necesario para corregir el error que existe entre el rumbo establecido por la trayectoria de la misión, y el rumbo real de la aeronave. Su estructura corresponde con un

controlador de tipo PD, con ganancia proporcional y derivativa. Con el fin de evitar que el controlador de rumbo genere cambios bruscos en la referencia del alabeo, se incluye también un filtro en la referencia generada, antes de entrar al lazo de control interno. Este filtro viene dado por la ecuación 5.49.

$$Filtro = \frac{1}{0.2 \cdot s + 1} \quad (5.49)$$

	K_p	K_i	K_d
Altura	-0.065	$-1.755 \cdot 10^{-3}$	0
Rumbo	0.361	0	2.500

Tabla 5.15. Ganancias de los controladores del lazo externo

Por último, el controlador de altura presenta una estructura de tipo PI, con ganancias proporcional e integral. Este generará el ángulo de referencia de cabeceo, de acuerdo a la diferencia entre la altura real de la aeronave y la altura de referencia según la trayectoria definida en la misión.

6 | Descripción y justificación detallada de la solución adoptada

En este capítulo se describe detalladamente la solución adoptada para la implementación de la simulación realista y la identificación de balizas con visión artificial. Así mismo, si bien el modelo dinámico y el sistema de control utilizados parten del trabajo realizado por Daniel Villalibre Vilariño, se han realizado modificaciones para adaptarlo a las herramientas proporcionadas por *UAV Toolbox*.

La arquitectura del sistema de simulación desarrollado es la dada por la Figura 6.1. Cada uno de estos subsistemas se explicará en las respectivas secciones de este capítulo. En el subsistema *Animación* se encuentra tanto la implementación del entorno de simulación realista, como el algoritmo de visión artificial, cada uno explicado en sus respectivas secciones.

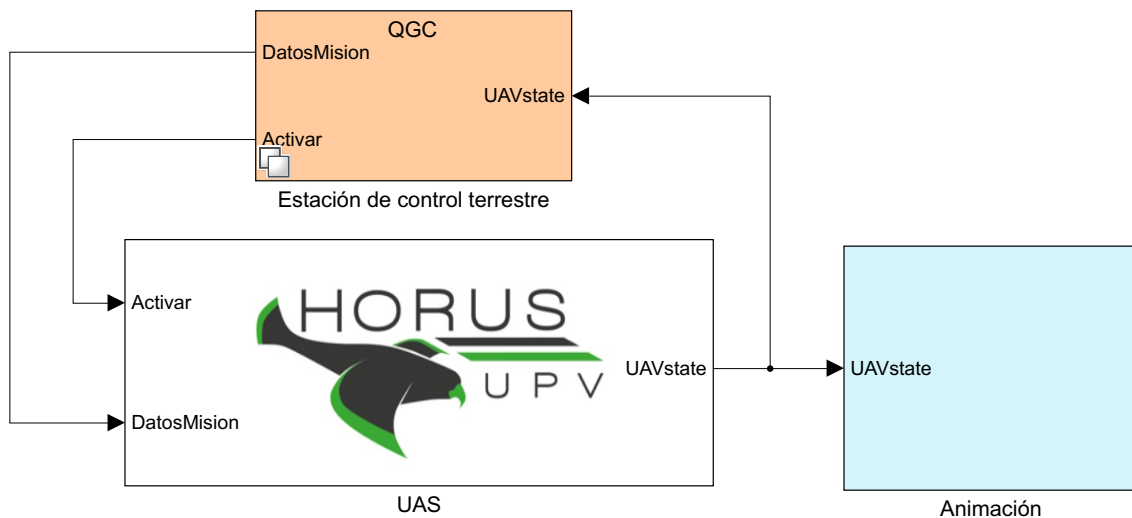


Figura 6.1. Esquema general de la implementación en Simulink

6.1. Implementación del sistema aéreo no tripulado

La implementación del modelo dinámico y el sistema de control se encuentra dentro del subsistema *UAS*, que se divide a su vez en los subsistemas del lazo de control externo, el lazo de control interno, el modelo dinámico y el entorno. En el subsistema *Entorno* se calculan las características atmosféricas necesarias para calcular las fuerzas y momento aerodinámicos (en concreto la densidad del aire), además de incluir el viento, cuyas componentes en ejes tierra se

modelan mediante ruido blanco y cuya magnitud depende de la potencia del ruido que se elija (por defecto es nula, pero se puede modificar según se desee incorporar el viento a la simulación). El esquema general puede observarse en la Figura 6.2.

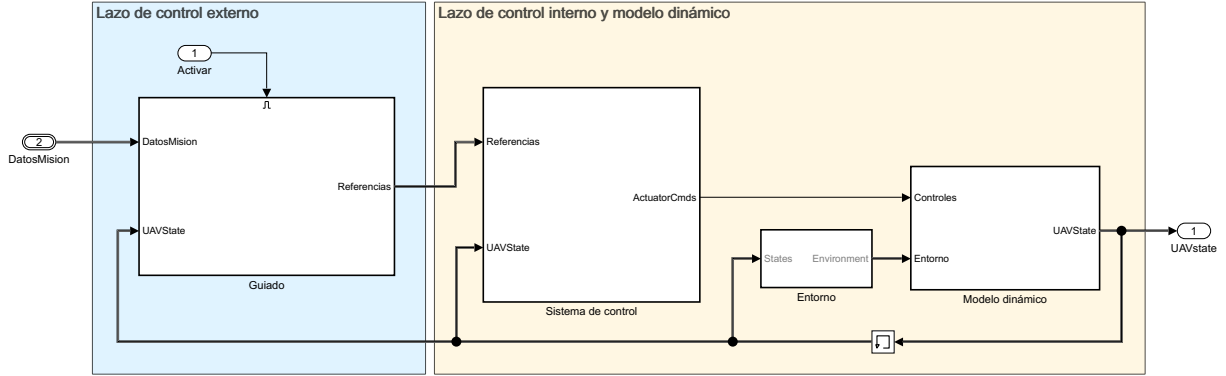


Figura 6.2. Esquema general del bloque *UAS*

6.1.1. Modelo dinámico

La base del modelo dinámico viene dado por las ecuaciones de Bryan vistas en el anterior capítulo, y originalmente estaban definidas con funciones de MATLAB[®] dentro de Simulink[®]. Sin embargo, la librería *Aerospace Blockset* proporciona un bloque para integrar las ecuaciones de un cuerpo rígido de 6 grados de libertad (*6 DoF Equations of Motion*), que permite seleccionar el tipo de representación de las ecuaciones (por ángulos de Euler o cuaterniones), así como el tipo de masa (fija o variable). El motivo principal por la elección de este bloque no es más que simplificar la representación del modelo dinámico, ya que la complejidad y los resultados obtenidos con este bloque son los mismos.

En concreto, las ecuaciones que implementa el bloque *6 DoF Equations of Motion*, según la documentación, corresponden a las ecuaciones 6.1, 6.2 y 6.3, donde $\vec{V}_b = [u \ v \ w]^T$, $\vec{\omega}_b = [p \ q \ r]^T$ e I es el tensor de inercia. Estas ecuaciones son iguales a las expuestas en el apartado de las ecuaciones de Bryan, solo que están expresadas en forma matricial. Se puede comprobar, calculando el producto de matrices y el producto vectorial, que estas ecuaciones son, de hecho, las mismas que las ecuaciones de traslación (ecuación 5.1 - 5.3), las ecuaciones de rotación (ecuación 5.6 - 5.8) y las ecuaciones que relacionan las velocidades angulares con las derivadas de los ángulos de Euler (ecuación 5.9 - 5.11).

$$\vec{F}_b = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m \cdot (\dot{\vec{V}}_b + \vec{\omega} \times \vec{V}_b) \quad (6.1)$$

$$\vec{M}_b = \begin{bmatrix} L \\ M \\ N \end{bmatrix} = I \cdot \dot{\vec{\omega}} + \vec{\omega} \times (I \cdot \vec{\omega}) \quad (6.2)$$

$$\begin{aligned}
 \begin{bmatrix} p \\ q \\ r \end{bmatrix} &= \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & \sin \phi \\ 0 & -\sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \\
 &+ \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & \sin \phi \\ 0 & -\sin \phi & \cos \phi \end{bmatrix} \begin{bmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \quad (6.3)
 \end{aligned}$$

La implementación final del modelo dinámico se da en la Figura 6.3. Como se puede observar, el bloque *6DoF Equations of Motion* proporciona la posición y velocidad de la aeronave en ejes tierra, los ángulos de Euler, la velocidad lineal y angular en ejes cuerpo, y la matriz de rotación de ejes tierra a ejes cuerpo. Esta matriz de rotación resulta de gran utilidad para calcular el vector gravedad en ejes cuerpo, ya que basta con premultiplicar el vector gravedad en ejes tierra ($[0 \ 0 \ g]^T$) con la matriz de rotación dada por el bloque. El subsistema *Bus setup* es necesario para generar la señal utilizada por todos los bloques del sistema de simulación, incluidos los de control, el de conexión con la estación terrestre y el de animación. La explicación en detalle de su definición y construcción se encuentra en el manual de usuario.

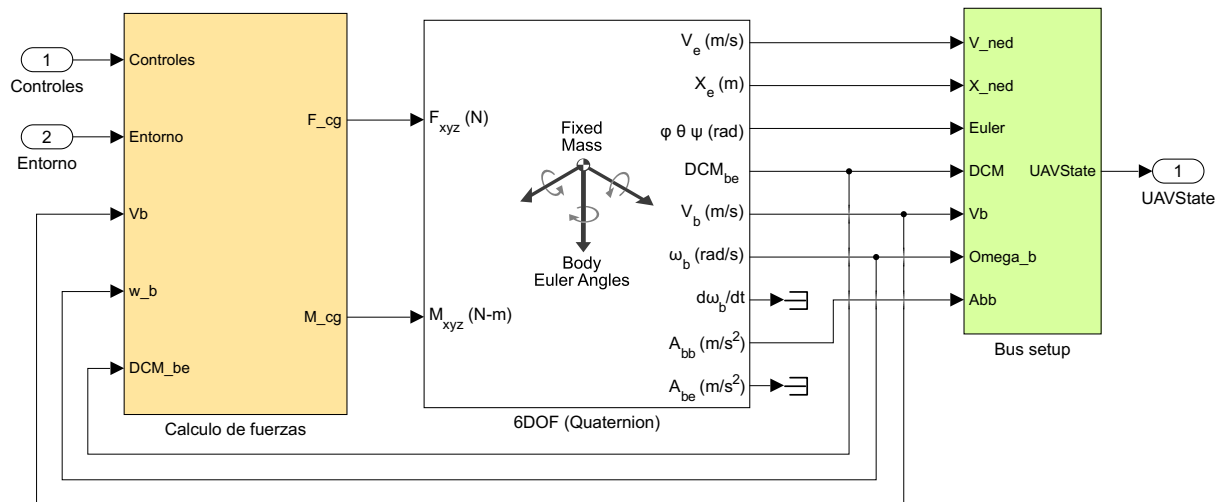


Figura 6.3. Implementación del modelo dinámico

Debido a que el bloque *6DoF Equations of Motion* acepta como parámetros de entrada la fuerza y momento total aplicado sobre el centro de gravedad de la aeronave, es necesario calcular la suma de todas las fuerzas primero. Esto se hace en el subsistema de cálculo de fuerzas, donde se implementan las ecuaciones vistas en la sección 5.1, “Modelado”. Esta implementación es la dada en la Figura 6.4. En el subsistema *Fuerza de la gravedad*, simplemente se multiplica el vector de gravedad en ejes tierra por la matriz de transformación de ejes tierra a ejes cuerpo, obteniendo así el vector gravedad en ejes cuerpo. Por otro lado, en el subsistema *Modelo aerodinámico* se implementan las ecuaciones del apartado 5.1.5 dentro de la sección 5.1, teniendo en cuenta el valor de las polares ($V, \alpha, \dot{\alpha}, \beta, \dot{\beta}$) calculado en el correspondiente subsistema. Por último, en el subsistema *Modelo propulsivo* se implementa el cálculo de las fuerzas y momentos asociados a la planta propulsiva, según las ecuaciones dadas en el apartado 5.1.4 dentro de la sección 5.1.

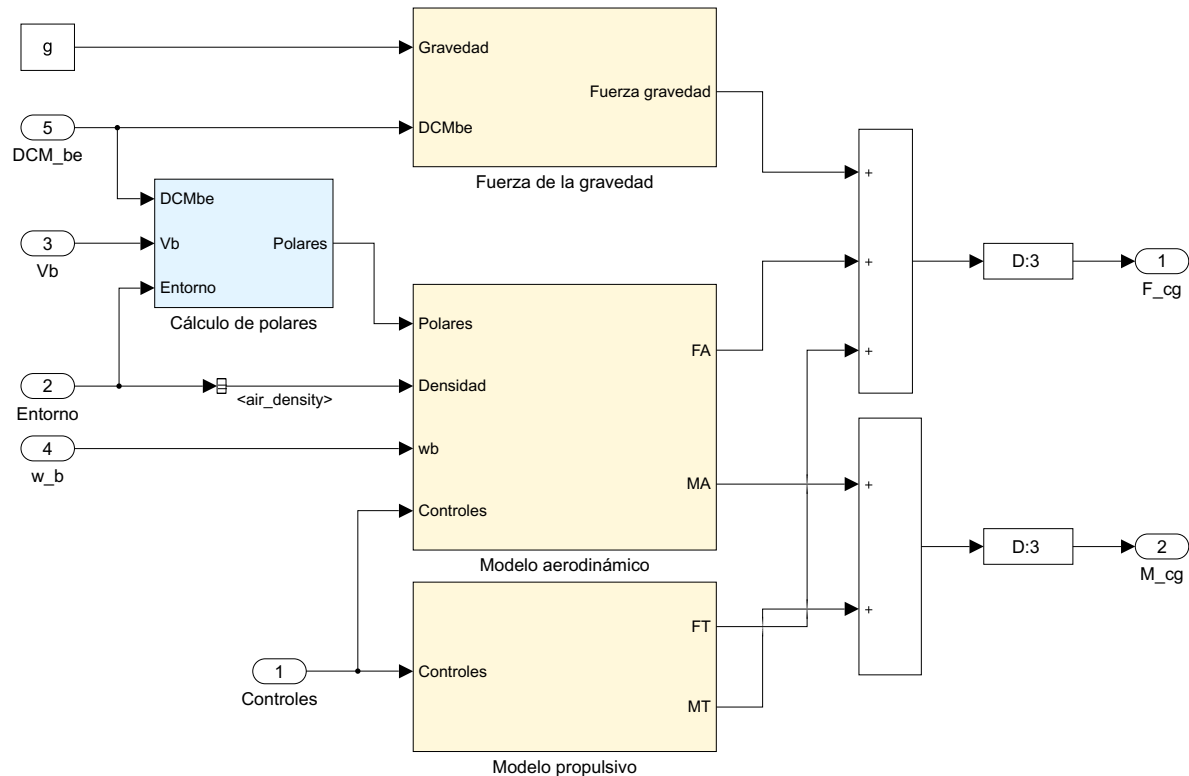


Figura 6.4. Implementación del cálculo de fuerzas y momentos totales

6.1.2. Sistema de control

Como se ha explicado en el capítulo 5, el sistema de control está compuesto de un lazo externo y uno interno, cuya representación se da en la Figura 5.1. La implementación del lazo interno es directa, como puede observarse en la Figura 6.5. En ella se aprecia el filtro utilizado para evitar cambios bruscos en la referencia del ángulo de alabeo, así como los controladores PID con las ganancias obtenidas en la sección 5.2. Cabe recordar que la salida de estos controladores está limitada según las características físicas del modelo considerado. Así mismo, estos generan los incrementos necesarios que deben aplicarse a las superficies de control para reducir a cero el error entre la referencia y el valor real, por lo que es necesario sumar los valores en las condiciones del punto de funcionamiento y así trabajar con valores absolutos.

La implementación del lazo de control externo es más compleja que la del control interno. En este caso se ha hecho uso de los bloques *Path Manager* y *Waypoint Follower* de MATLAB[®], que proporcionan algoritmos de guiado. Se utilizan estos bloques por diversas razones:

1. Ofrecen algoritmos de guiado ya programados, con suficiente complejidad dada la naturaleza de este trabajo [15].
2. Permiten modificar parámetros del algoritmo de guiado con mayor facilidad, permitiendo su optimización al usuario del sistema de simulación desarrollado en este trabajo.
3. Pueden simularse utilizando generación de código. De esta forma, si bien el tiempo de compilación es mayor, la ejecución de la simulación es más rápida.

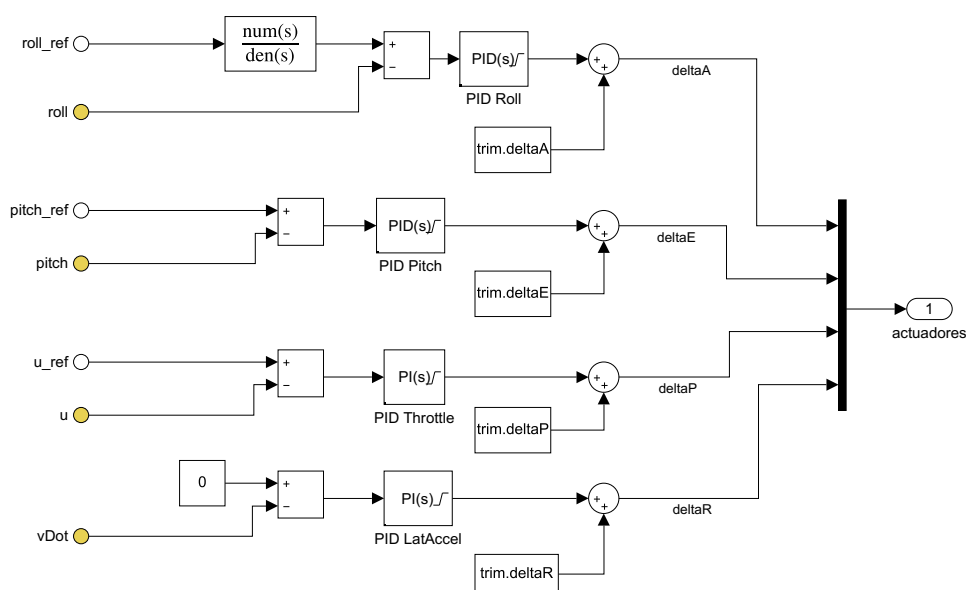


Figura 6.5. Implementación de los controladores del lazo de control interno

Path manager [16]

Este bloque calcula los parámetros de la misión cambiando secuencialmente entre los puntos de la misión dados por la estación de control terrestre. En concreto, devuelve dos waypoints consecutivos, el actual (al que se dirige la aeronave) y el previo, necesarios para el cómputo de la trayectoria nominal y el cálculo de las referencias, como puede observarse en la Figura 6.6. Mediante un cambio en el parámetro de entrada *isModeDone*, indicando que la aeronave ha llegado al waypoint de destino, el bloque devuelve el siguiente waypoint, hasta terminar la misión.

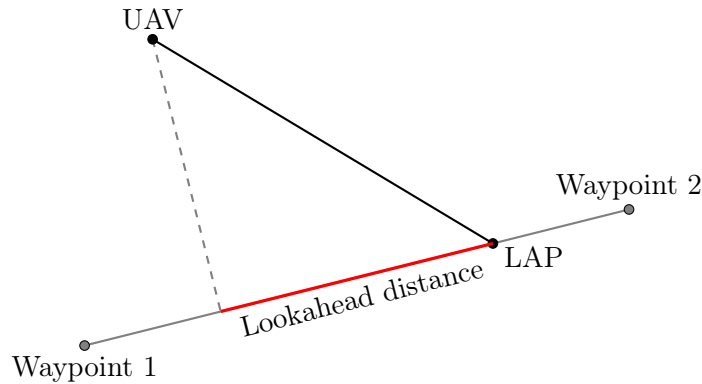
De igual forma, permite cambiar el tipo de misión, ya sea ejecutando la misión desde el primer hasta el último waypoint, realizar una espera en el waypoint actual (para aeronaves de ala fija se especifica un radio de giro alrededor del punto de espera, denominado *loiter radius*), repetir la misión una vez alcanzado el último waypoint o retornar a la posición de despegue. Para el caso de estudio que concierne a este trabajo, solo se hará uso del modo por defecto, en el que ejecuta la misión desde el primer al último waypoint. Se deja la posibilidad no obstante, de modificar este parámetro para realizar distintas misiones.

Waypoint Follower [17]

Este bloque proporciona el algoritmo de navegación utilizado en el sistema de simulación. En concreto, se trata de un algoritmo de guiado no lineal para el seguimiento de trayectorias. Los parámetros de entrada a este bloque, dados en la Figura 6.6, son:

1. *Pose*: corresponde a la posición de la aeronave en un instante dado, expresada en ejes tierra (NED). También incluye el rumbo de la aeronave, calculado de acuerdo a la ecuación 5.36.
2. *Waypoints*: corresponde a la información de los waypoints proporcionada por el bloque **Path Manager**, a partir de la cual, junto a la posición de la aeronave, calcula los parámetros de salida.

3. *LookaheadDistance*: la distancia sobre la trayectoria del punto al que debe dirigirse la aeronave (LAP), respecto del punto de intersección del vector normal de la recta que pasa por la posición de la aeronave. Esto se visualiza mejor en la siguiente gráfica:



La elección de este parámetro influye en la respuesta ante cambios en la trayectoria. Un valor elevado incrementa el tiempo necesario para que la aeronave reduzca el error entre la posición real de la aeronave y la trayectoria definida por los waypoints (denominado *cross-track error*). Por otro lado, un valor pequeño produce cambios bruscos en la referencia del ángulo de alabeo. La elección de un valor óptimo correspondería a la optimización de los algoritmos de guiado y el sistema de control, que está fuera del alcance de este trabajo. Por lo tanto, se ha elegido un valor de partida de 10 m, ya que produce una respuesta razonable.

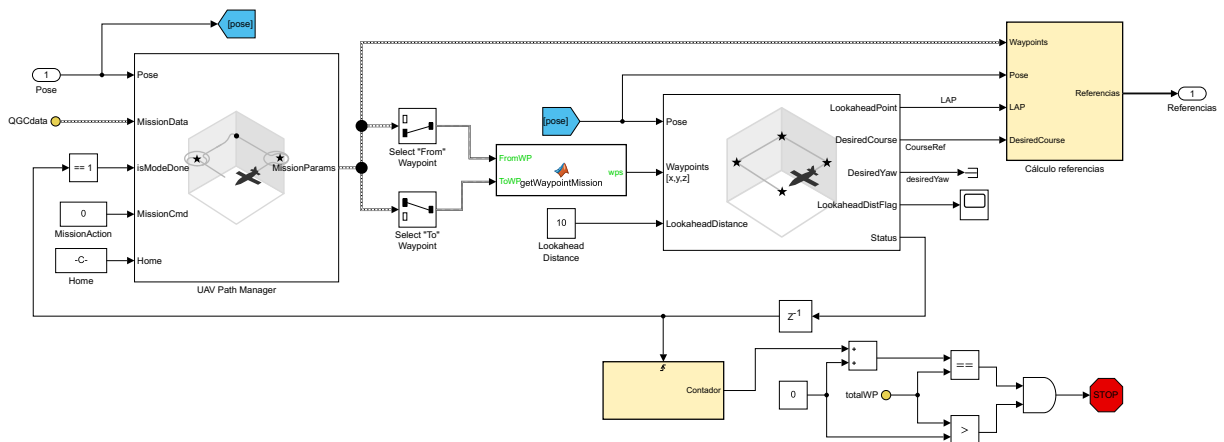


Figura 6.6. Implementación del bloque de guiado

Los parámetros de salida del bloque **Waypoint Follower**, utilizados para el cálculo de referencias, son los siguientes:

1. *LookaheadPoint* (LAP): coordenadas del punto de referencia sobre la trayectoria, especificado en ejes tierra.
2. *DesiredCourse*: rumbo que debe seguir la aeronave para alcanzar el punto de referencia. Se define como el ángulo de la dirección del vector velocidad (proyectado en el plano XY) respecto al Norte.
3. *LookaheadDistFlag*: indica si se produce una saturación de la distancia mínima.
4. *Status*: indica que la aeronave ha alcanzado el waypoint de destino. Esta señal es utilizada por el bloque **Path Manager** para cambiar entre los puntos de la misión.

Cálculo de las referencias del lazo de control externo

El cálculo de las referencias utilizadas posteriormente por el lazo de control interno se realiza en el subsistema *Cálculo de referencias*. Recordando el lazo de control externo explicado en la sección 5.2, las referencias generadas corresponden al ángulo de alabeo, ángulo de cabeceo, aceleración lateral y velocidad. Teniendo en cuenta los parámetros devueltos por el bloque **Waypoint Follower**, el cálculo de las referencias, según la implementación de la Figura 6.7, se realiza de acuerdo a los siguientes criterios:

1. Referencia del ángulo de cabeceo: se calcula mediante el controlador PI obtenido en la sección 5.2, a partir del error en la altura. Este error viene dado por la diferencia entre la altura de referencia, dada por la coordenada z del LAP, y la altura real de la aeronave.
2. Referencia del ángulo de alabeo: obtenida mediante el controlador PD, dado en la sección 5.2, a partir de la diferencia entre el rumbo deseado y el rumbo real de la aeronave.
3. Referencia de la velocidad: el cálculo de la velocidad de referencia se realiza mediante una interpolación lineal entre la velocidad de referencia en los dos waypoints consecutivos, dada en la ecuación 6.4, donde A es el waypoint previo, B es el punto de referencia en la trayectoria (LAP) y C es el waypoint de destino.

$$u_{ref} = u_A + \frac{u_C - u_A}{|\vec{AC}|} \cdot |\vec{AB}| \quad (6.4)$$

4. Referencia de la aceleración lateral: no se requiere ningún cálculo, puesto que siempre es igual a cero. Esto se incluye directamente en el lazo de control interno, como puede observarse en la Figura 6.5.

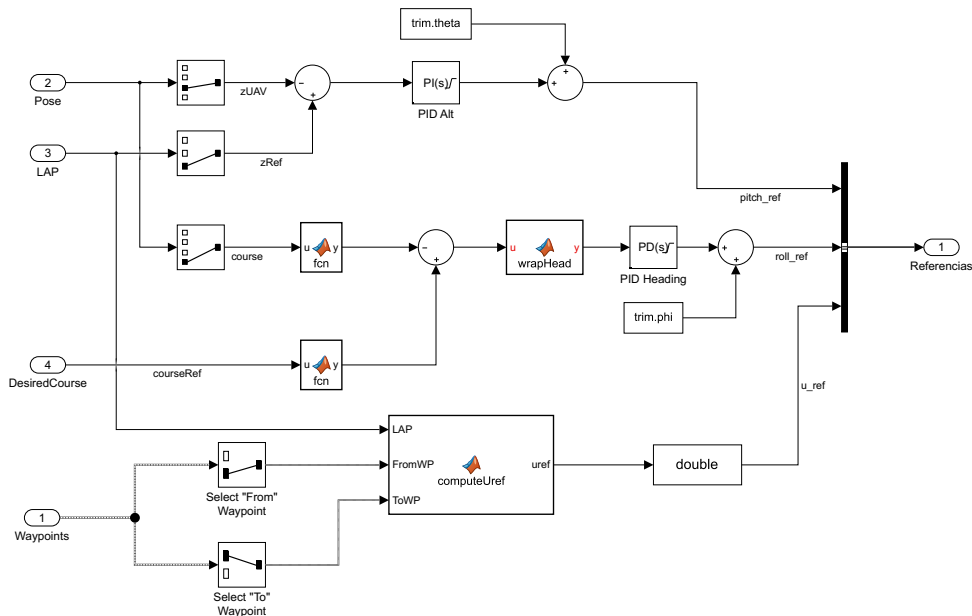


Figura 6.7. Cálculo de las referencias utilizadas por el lazo de control interno

6.2. Estación de control terrestre

La estación terrestre utilizada para establecer la conexión con Simulink[®] es *QGroundControl*, en adelante QGC. Esta estación de control terrestre está disponible de forma gratuita, y permite el control del vuelo y la planificación de misiones autónomas para sistemas aéreos no tripulados. Utiliza el protocolo de comunicación MAVLink, que se caracteriza por ser un protocolo ligero de comunicación con drones, o entre componentes a bordo del mismo dron.

6.2.1. Definición de la misión

Mediante QGC se puede definir una serie de misiones, con sus respectivos waypoints indicando la trayectoria que debe seguir la aeronave. En concreto, la misión utilizada en este trabajo es la dada en la Figura 6.8.

Cada waypoint está definido por su latitud, longitud, y altitud. Estos datos son los enviados a MATLAB[®] para realizar la conversión a ejes tierra (coordenadas NED) y definir los parámetros utilizados por el lazo de control externo. Esta misión es la utilizada para validar el sistema, e incluye todas las balizas creadas y explicadas en la sección 6.4. Además, mediante QGC se definen otras dos misiones para extraer los datos del terreno utilizados en el algoritmo de visión artificial.

El uso de QGC permite realizar una simulación más realista de la misión, ya que esta estación de tierra será utilizada posteriormente en pruebas de vuelo, una vez finalizadas las simulaciones necesarias para verificar el sistema de control diseñado.



Figura 6.8. Misión definida en QGC

6.2.2. Conexión con *Simulink*

La conexión con Simulink[®] se realiza de acuerdo a la estructura de la Figura 6.9. La interfaz de comunicación con MAVLink se obtiene del proyecto *UAV Package Delivery*, proporcionado como ejemplo en *UAV Toolbox*. Esta función devuelve los parámetros de la misión, donde se encuentra la información sobre los waypoints en coordenadas LLA (Latitud-Longitud-Altitud), el número de waypoints, utilizado por el lazo de control externo para determinar la finalización de la misión, y un valor de tipo lógico indicando que se ha subido correctamente la misión, lo cual es utilizado para activar el lazo de control externo.

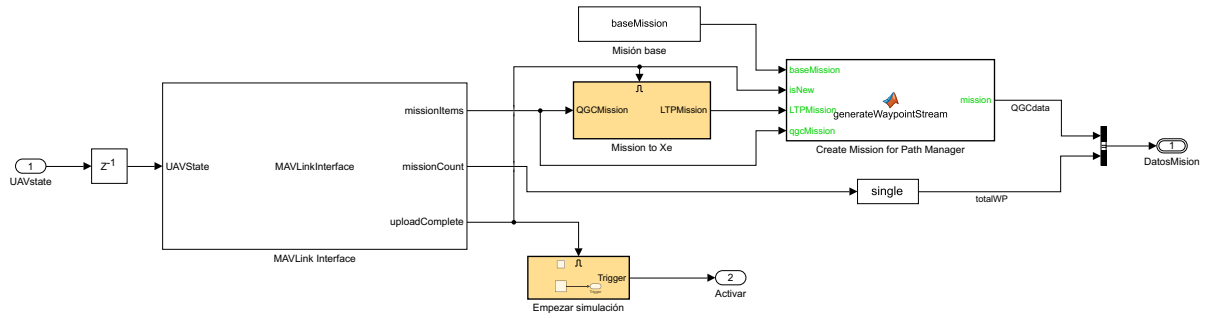


Figura 6.9. Conexión con QGC en Simulink

6.3. Entorno de simulación realista

El entorno de simulación realista es la base para incorporar los algoritmos de visión artificial del sistema de simulación. No solo permite visualizar la respuesta de la aeronave a lo largo de la trayectoria definida, sino que permite obtener datos del entorno mediante sensores. Por lo tanto, el paso previo a la implementación de los algoritmos de visión artificial corresponde con la creación del escenario en el que se simulará la misión. En esta sección se explicará la solución adoptada para establecer la conexión de Simulink[®], así como el proceso de creación del escenario en Unreal Engine[®] (en adelante Unreal).

6.3.1. Implementación en Simulink

La implementación en Simulink[®] consta principalmente de los siguientes 3 bloques, dados en la Figura 6.11, pertenecientes a *UAV Toolbox*:

1. Simulation 3D Scene Configuration: en este bloque se configuran los parámetros de la escena, como la perspectiva de visualización (situada en el origen del escenario o ligada al vehículo), el tiempo de muestreo y la fuente del escenario. Existen tres opciones para la elección del escenario, la primera es mediante los escenarios que vienen instalados por defecto en *UAV Toolbox*, la segunda es mediante una co-simulación con *Unreal Editor*, y la tercera es mediante un archivo ejecutable que contenga el escenario. Para poder trabajar con un escenario propio, es necesario ejecutar la simulación a partir de un archivo ejecutable o en conjunto con *Unreal Editor*. En este trabajo se elige la opción de simulación a partir del ejecutable, ya que la simulación es más rápida en comparación con la simulación conjunta en *Unreal Editor*.
2. Simulation 3D UAV Vehicle: configura el tipo de vehículo visualizado en la simulación. Para ello, los datos de entrada corresponden a la traslación y rotación de la aeronave, obtenidas del modelo dinámico. Cabe destacar que el sistema de coordenadas utilizado por Unreal es distinto al utilizado para representar la posición y orientación por el modelo dinámico. Es por esto que es necesario convertir los datos al sistema de coordenadas del entorno de visualización, que se realiza en el subsistema *Prepare UAV Data*. Por otra parte, en la versión de MATLAB[®] en la que se ha desarrollado el sistema de simulación no es posible modificar las características geométricas del vehículo, simplemente su color y tipo (ala fija o ala rotatoria).
3. Simulation 3D Camera: en este bloque se configuran las características del modelo de cámara introducido en la simulación. Los parámetros que lo definen son la posición y orientación

del anclaje a la aeronave, así como las características ópticas de la cámara como la distancia focal, la resolución, o el centro óptico. Los datos proporcionados por este bloque serán los analizados por el algoritmo de visión artificial para detectar las balizas sobre el terreno.

Para comprender mejor el funcionamiento de estos bloques, en la Figura 6.10 se representa la comunicación entre Simulink[®] y Unreal . La obtención de datos del entorno se realiza mediante el modelo de cámara proporcionado por el bloque *Simulation 3D Camera*, mientras que la representación de la aeronave viene dada por el bloque *Simulation 3D UAV Vehicle*. Por último, el bloque *Simulation 3D Scene Configuration* es la base de la simulación, y no puede ser omitido, a diferencia de bloque que simula una cámara real.

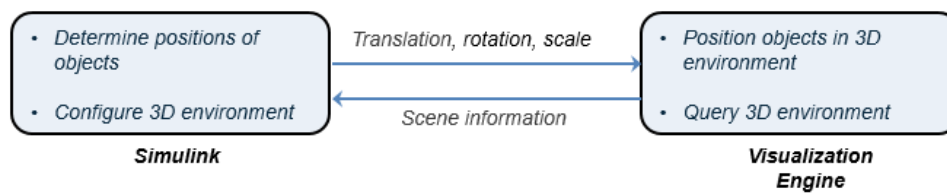


Figura 6.10. Comunicación con el entorno de simulación 3D [6]

Teniendo en cuenta lo anteriormente explicado, el orden de ejecución de los bloques durante la simulación viene dado por la prioridad asignadas a los mismos. En concreto, la prioridad se asigna de -1 a $+1$, determinando el siguiente orden de ejecución:

1. El bloque *Simulation 3D Scene Configuration*, cuya prioridad es igual a -1 , es el primero en ejecutarse. Este inicializa el vehículo y envía los datos de la traslación y rotación al bloque *Simulation 3D Scene Configuration*.
2. El bloque *Simulation 3D Scene Configuration*, siendo el segundo en ejecutarse (prioridad igual a 0), recibe la información de la posición y orientación del vehículo y envía estos datos a los bloques de sensores.
3. Los bloques de sensores, siendo los últimos en ejecutarse (prioridad $+1$), obtienen los datos del entorno de forma precisa.

Finalmente, la implementación de estos bloques puede observarse en la Figura 6.11. En el siguiente apartado se explicará con detenimiento el proceso para crear el escenario realista utilizado en el bloque *Simulation 3D Scene Configuration*.

6.3.2. Creación del escenario realista

El escenario realista creado en este trabajo se realiza con Unreal Editor, el editor que permite crear proyectos de diversa índole para el motor de juegos Unreal. Si bien en sus inicios estaba orientado a juegos, en la actualidad sus aplicaciones han aumentado. Un ejemplo de ello es la creación de tomas cinematográficas, ya sea en terrenos realistas o en mundos ficticios.

Teniendo en cuenta que el sistema de simulación desarrollado en este trabajo será utilizado para simular la misión del concurso *UAS Challenge*, es de interés disponer de un entorno de visualización con el terreno en el que tendrá lugar la misma. Por este motivo, es necesario disponer de datos sobre la elevación del terreno e imágenes para reconstruir el mismo [18]. A continuación se explicará el procedimiento seguido para la obtención de cada uno de ellos.

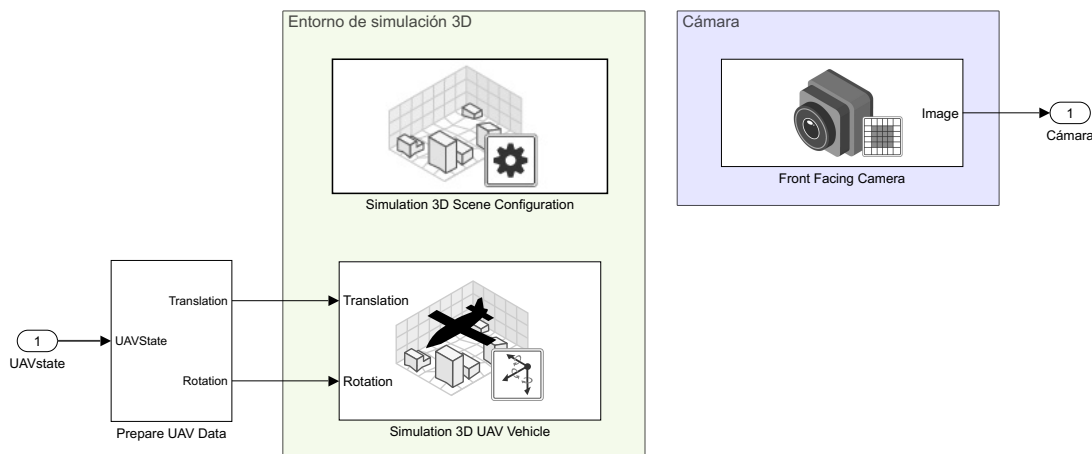


Figura 6.11. Implementación de la comunicación con el entorno realista

Obtención del Modelo Digital del Terreno (MDT)

La información sobre la elevación del terreno viene dada por modelos digitales de elevación, que son representaciones matemáticas de la altura del terreno u otras estructuras respecto al nivel del mar. Los valores están contenidos en archivos de formato ráster, para permitir su análisis en Sistemas de Información Geográfica (GIS por sus siglas en inglés). El Modelo Digital de Elevación (MDE) puede ser de dos tipos [19]:

- Modelo Digital del Terreno (MDT): recrea la forma del terreno, sin incluir aquellos elementos que no forman parte del terreno. Es decir, no incluye edificaciones, infraestructuras o la propia vegetación.
- Modelo Digital de Superficies (MDS): representa todos los elementos existentes en la superficie terrestre, como la vegetación, las edificaciones, infraestructuras y el propio terreno.

Considerando que en este trabajo se desea recrear el terreno en el que se desarrolla el concurso, el MDE más adecuado sería el MDT. Según las fuentes consultadas, la resolución que se puede obtener del MDT es mayor o menor. Existen diversas fuentes con datos del MDT a nivel global, típicamente con una resolución de 30 m. A falta de datos más precisos, se puede generar el terreno a partir de estos datos, aunque se requerirá de mayor tratamiento una vez importado al editor de Unreal, debido a imperfecciones que puedan aparecer en la superficie por la resolución. Algunas fuentes gratuitas para disponer de datos MDT a escala global son [20]:

- SRTM (Space Shuttle Radar Topography Mission): elaborado por la NASA en el año 2000, y dispuesto al público en el 2014, contiene modelos digitales del terreno de hasta 30 m de resolución. Se utilizó la técnica conocida como *Interferometric Synthetic Aperture Radar (InSAR)* para recoger los datos suficientes para generar el MDT. Estos datos están disponibles en *USGS Earth Explorer*.
- ASTER Global Digital Elevation Model: fruto de la colaboración conjunta de la NASA y Japón, proporciona modelos digitales de elevación con resolución de 90 m para todo el mundo, y de 30 m para Estados Unidos. Los datos se obtuvieron mediante la técnica de fotogrametría y están disponibles también en *USGS Earth Explorer*.
- JAXA's Global ALOS 3D World: es un MDS con resolución de 30 m, disponible gratuitamente en el portal de JAXA (*JAXA Global ALOS Portal*).

Además de la página web de *USGS Earth Explorer*, existen otras herramientas como *OpenStreetMap*, *Open Topography*, *Sentinel Satellite Data* o *NASA Earth Observations (NEO)*. Sin embargo, el MDT finalmente utilizado para crear el escenario proviene de la página web oficial del gobierno de Reino Unido [21]. En este caso los datos provienen de un mapeo con la técnica LiDAR (*Light Detection and Ranging*), ofreciendo una resolución de 1 m, motivo por el cual se ha elegido como MDT para construir el escenario.

Los MDT proporcionados por técnicas LiDAR tienen la desventaja de no estar disponibles para toda la superficie de Inglaterra. En concreto, la zona del concurso solo estaba cubierta en un 50 % aproximadamente, por lo que se optó por elegir otra localización cercana. La división del terreno se realiza por hojas o celdas, y la utilizada en este trabajo corresponde con la SK82NW, descargada de la página web dada en la referencia [21]. Se decidió no representar la zona del concurso, sino otra cercana, para aprovechar la resolución de 1 m que proporcionaba este MDT. No obstante, esta decisión no afecta a la simulación de la misión, ya que el entorno es similar a la zona del concurso (en ambos casos son zonas de cultivo).

Una vez obtenido el MDT, es necesario tratar este archivo para adecuarlo a los formatos admitidos por Unreal Editor. Para ello, es necesario el uso de un Sistema de Información Geográfica (GIS). En este caso se utiliza QGIS, ya que es software libre y de código abierto. Mediante QGIS se recorta el MDT para que tenga una longitud de 2017x2017 m, debido al tamaño recomendado para los mapas en Unreal.

Durante la creación de un mapa en Unreal, según la documentación disponible [8], es necesario especificar los siguientes parámetros:

1. Número de componentes: los componentes son la unidad básica de renderizado, cálculo de visibilidad y de colisión. Cada componente es un cuadrado y el mapa está compuesto por un número finito de ellos.
2. Número de secciones: unidades de división de los componentes, que son la unidad base para el cálculo del nivel de detalle del mapa (*LOD - Level Of Detail*). Los componentes pueden estar divididos en 1 o 4 secciones (2x2).
3. Número de Quads: unidad de división de las secciones.

La elección de estos parámetros debe hacerse teniendo en cuenta el rendimiento. Componentes de menor tamaño incrementan el detalle del mapa pero disminuye el rendimiento. Esto se debe a que cada componente incurre un coste de procesamiento en la CPU, por lo que un menor número de componentes consume menos recursos de la CPU para el renderizado. Debido a que el sistema de simulación se desarrolla en un ordenador con recursos limitados, el número de componentes debe mantenerse a un mínimo, por lo que se utilizarán las dimensiones recomendadas proporcionadas en la documentación.

En la Tabla 6.1 se proporcionan algunas de las dimensiones recomendadas para el mapa. Teniendo en cuenta que el número de vértices indica la resolución del mapa (píxeles), y que cada píxel corresponde a 1 m, se opta por una resolución de 2017x2017, ya que es menor que la distancia cubierta por el MDT descargado y proporciona un mapa lo suficientemente grande como para tener suficiente terreno en el que realizar la simulación.

Finalmente, una vez determinada la resolución que debe tener el archivo importado a Unreal, se recorta el MDT mediante QGIS para que tenga dicha dimensión. Este archivo está en formato GeoTIFF, que no es compatible con los archivos aceptados por Unreal. Para ello, es necesario

Tamaño total (vértices)	Quads/ Sección	Secciones/ Componente	Tamaño del componente	Número total de componentes
8129x8129	127	4 (2x2)	254x254	1024 (32x32)
4033x4033	63	4 (2x2)	126x126	1024 (32x32)
2017x2017	63	4 (2x2)	126x126	256 (16x16)
1009x1009	63	4 (2x2)	126x126	64 (8x8)

Tabla 6.1. Tamaños del mapa recomendados para Unreal Engine [8]

exportar el MDT como un mapa de altura (en inglés denominado *heightmap*), en formato PNG, donde cada píxel almacena un valor de elevación en escala de grises, siendo el blanco la máxima elevación y el negro la mínima elevación. Afortunadamente, existe un plugin en QGIS desarrollado por la comunidad, que permite exportar de formato GeoTIFF a formato PNG. Tras este proceso, los datos del terreno pueden importarse en Unreal.

Obtención de las imágenes satelitales

La imagen satelital es necesaria para completar la información proporcionada por el MDT. Si bien no es necesario incorporar una imagen satelital al escenario creado en Unreal, la adición de esta proporciona una base para conseguir que el entorno simulado sea lo más realista posible. Por ello, se puede utilizar la imagen simplemente como guía para determinar qué texturas y objetos introducir en el mapa. Como segunda opción, se puede utilizar la imagen satelital como textura del terreno, sin realizar modificaciones posteriores. Para ello, la imagen satelital debe cumplir los siguientes requisitos:

- Debe tener alta resolución, de forma que el escenario generado tenga suficiente calidad.
- Debe estar georreferenciada, para poder trabajar con ella en QGIS.

Aunque existen diversas fuentes con imágenes satelitales de forma gratuita, entre ellas *USGS Earth Explorer*, *Landviewer* o *Copernicus Open Access Hub*, se opta por utilizar las imágenes proporcionadas por Google Earth. El factor determinante para tomar esta decisión es la facilidad para el tratamiento de estas imágenes, ya que pueden ser importadas directamente desde QGIS. El procedimiento para obtener la imagen deseada es entonces:

1. Importar las imágenes satelitales de Google a QGIS.
2. Recortar la imagen satelital de acuerdo a la capa MDT obtenida con anterioridad. De esta forma, ambas capas coinciden y la imagen del satélite se corresponde con los datos de elevación.
3. Importar la imagen satelital ya recortada a un editor de imágenes. Este paso es necesario ya que la imagen satelital con la que trabaja QGIS está en formato GeoTIFF, pero este no es compatible con Unreal. En este caso, se puede realizar mediante GIMP, un editor de imágenes gratuito.
4. Exportar la imagen satelital a formato PNG, de modo que pueda ser incluida en Unreal como una textura.

Construcción del escenario en Unreal Editor

Una vez se obtiene el Modelo Digital del Terreno y la imagen satelital en formato PNG, es posible crear el escenario. El resultado de los procedimientos anteriores se da en la Figura 6.12. El MDT se utiliza para importar los datos del terreno en la herramienta de creación de paisajes de Unreal. Por otro lado, la imagen satelital se utiliza para dar una textura al terreno.

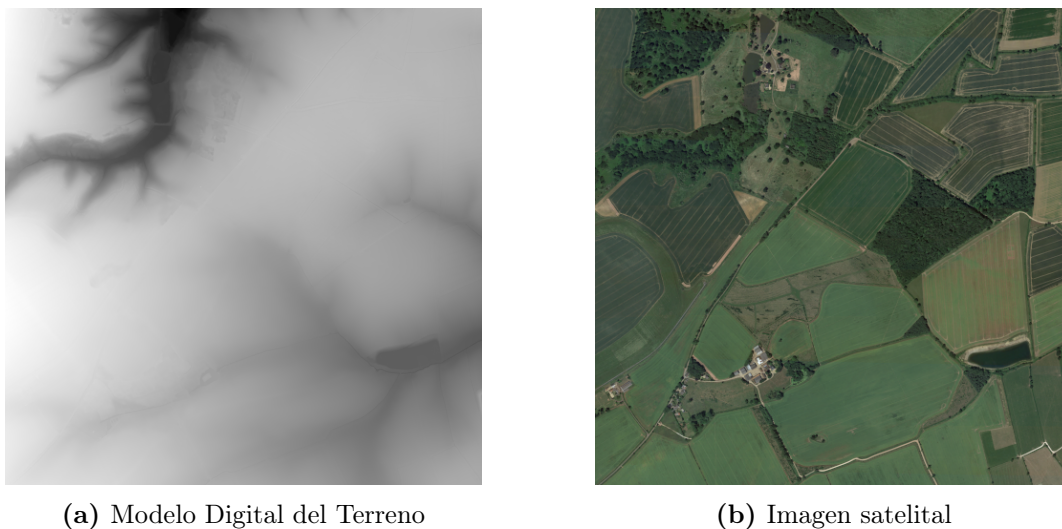


Figura 6.12. Archivos necesarios para la creación del escenario realista

El resultado del proceso de importación a Unreal, explicado con detenimiento en el manual de usuario [22], es el de la Figura 6.13. Como se puede observar, se consigue un resultado satisfactorio, pudiéndose determinar las estructuras y la vegetación o los cultivos en cada zona. Sobre este terreno serán incluidas las balizas para su posterior identificación con visión artificial.

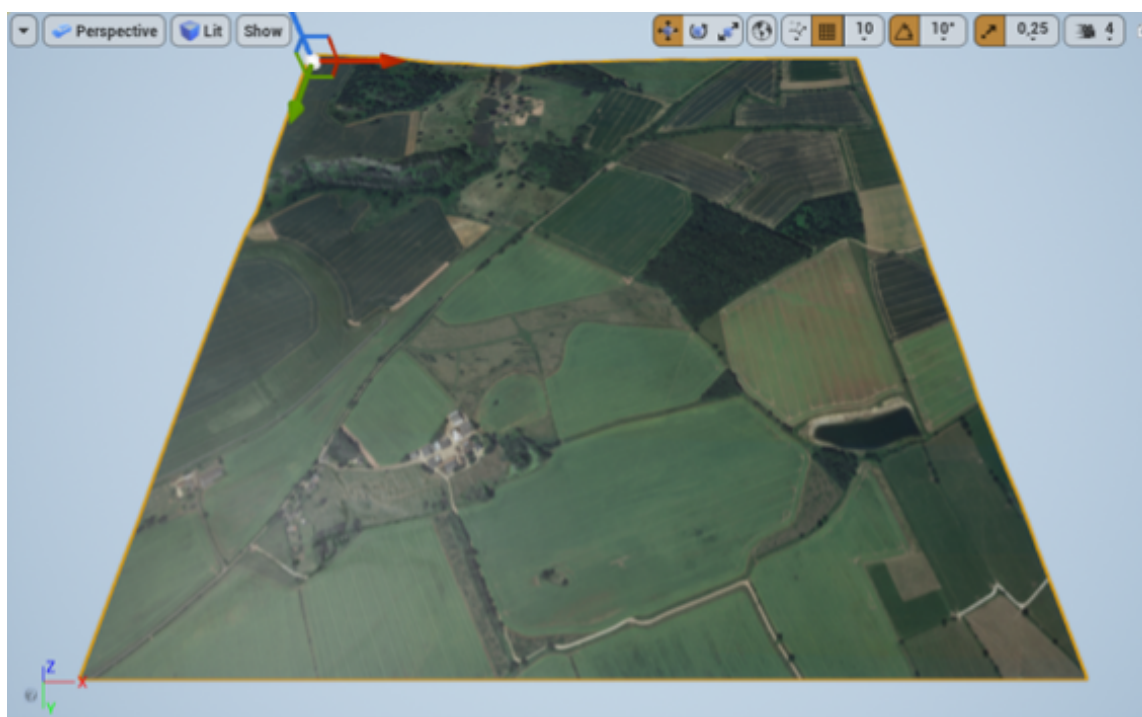


Figura 6.13. Resultado final del escenario en Unreal Engine

El resultado obtenido de por sí tiene una resolución aceptable, pero no representa fielmente la realidad, ya que es una textura en dos dimensiones, mientras que los árboles, la vegetación y las casas son tridimensionales. En caso de que se desee un mayor realismo en la simulación, existen diversas bases de datos con texturas y objetos, como son:

- Quixel Megascans: es una librería en línea que reúne miles de recursos de fotogrametría, en dos o tres dimensiones, para crear entornos ultra realistas. Esta colección dispone de texturas y objetos que permiten recrear entornos reales en Unreal. Además, está disponible de forma gratuita para todos los usuarios.
- Bazar de Epic Games: es una colección, dentro del programa *Epic Games Launcher*, de usuarios que crean escenarios y objetos para ser utilizados en proyectos propios. Sin embargo, solo unos pocos están disponibles de forma gratuita, en la sección de colecciones permanentemente gratis.

Es importante recalcar que, cuánto más realista se haga el escenario, más recursos computacionales requerirá la simulación. Dadas las limitaciones del ordenador en el que se ha desarrollado este trabajo, se opta por no incluir los detalles del terreno en tres dimensiones. Además, este proceso requiere de una dedicación considerable de tiempo, lo cual no es viable en el plazo de desarrollo del trabajo. Por estos motivos, el escenario realista se limita a incluir la imagen satelital como única textura, dejando la opción de incluir con detalle otras características para futuros trabajos.

6.4. Visión artificial

En esta sección se explica el proceso seguido para implementar algoritmos básicos de detección de objetos para la identificación de las balizas, mediante visión artificial. La solución adoptada consiste en utilizar modelos ya programados, en los cuales solo es necesario su entrenamiento a partir de imágenes obtenidas durante la simulación. Esta decisión se toma debido a las restricciones temporales para la realización de este proyecto, reduciéndose así el tiempo necesario para la construcción del modelo para detectar las balizas.

6.4.1. Balizas

Según la normativa del concurso, las balizas terrestres están compuestas de un cuadrado externo de color blanco, con longitud de 2 m. En el interior del mismo se encuentra un segundo cuadrado de color, con longitud de 1 m, dentro del cual se encuentra un carácter, que puede ser una letra de la A a la Z en el alfabeto inglés, o un número del 1 al 9. En la Figura 6.14a puede observarse la estructura de las balizas que se utilizarán en el concurso, mientras que en la Figura 6.14b puede observarse la baliza creada en Unreal. Conviene recordar que, si bien las balizas del concurso incluyen un carácter alfanumérico, en este proyecto se considera una primera aproximación en la que no se identifica el carácter, por lo que se omite en la representación.

La baliza se crea como un objeto que puede ser importado en el mapa, de modo que puedan ser visualizadas por la cámara durante la simulación. De igual forma, la localización de las mismas puede obtenerse de las propiedades del objeto, lo cual será utilizado para validar la precisión del algoritmo de detección. Cabe destacar que el color de las balizas creadas en Unreal varía, ya que así se consideran más opciones.

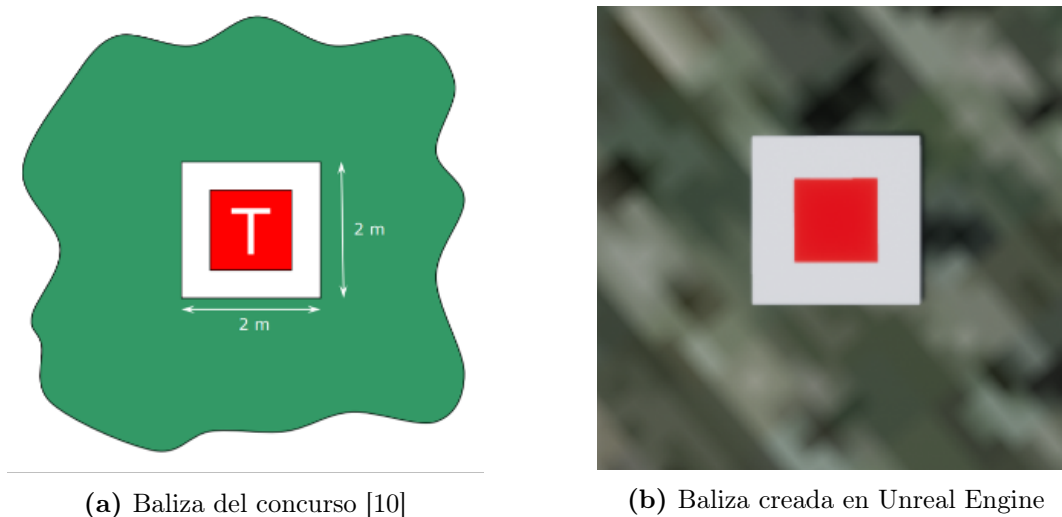


Figura 6.14. Balizas utilizadas en la simulación

6.4.2. Algoritmo de detección de balizas

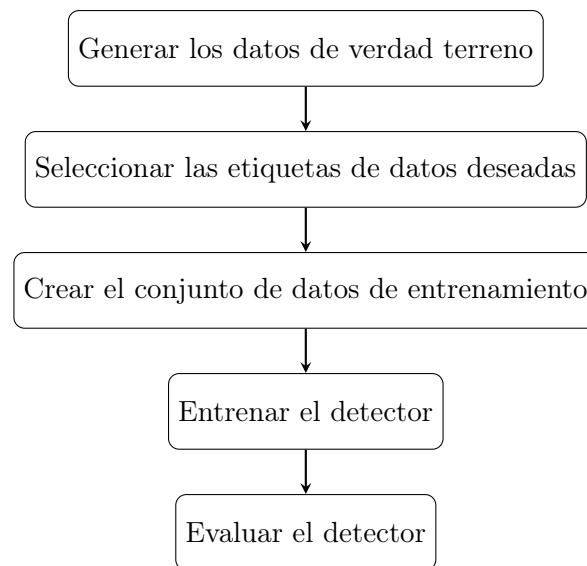
La detección de objetos es una técnica de visión artificial que permite localizar instancias de un objeto en un vídeo o una imagen. Los algoritmos de detección de objetos se implementan habitualmente mediante aprendizaje automático (*Machine Learning*) o *Deep Learning*. El aprendizaje automático utiliza algoritmos de regresión o árboles de decisión, mientras que el *Deep Learning* utiliza redes neuronales que se asemejan a las conexiones neuronales biológicas de nuestro cerebro.

El tipo de aprendizaje de ambas puede ser supervisado o no supervisado. En el aprendizaje supervisado, deben proporcionarse datos de entrenamiento, en los que, para cada entrada, se identifica el valor esperado a la salida. De esta forma, el algoritmo es entrenado para producir las salidas deseadas antes unas condiciones de entrada dadas. Se denomina aprendizaje supervisado ya que requiere de intervención humana, etiquetando el conjunto de datos utilizados. Por otro lado, en el aprendizaje no supervisado se proporciona un conjunto de datos sin etiquetar, de modo que es el propio algoritmo el que extrae patrones y relaciones entre ellos.

Los algoritmos de *Deep Learning*, sean o no entrenados de forma supervisada, requieren una gran cantidad de datos para producir resultados satisfactorios, además de una capacidad computacional elevada. Teniendo en cuenta la naturaleza de este trabajo, se decide finalmente implementar un algoritmo de detección de objetos mediante aprendizaje automático, ya que el objetivo es demostrar la capacidad de integración de algoritmos de visión artificial en la simulación, no siendo objeto de estudio el algoritmo concreto utilizado.

Dentro de los tipos de algoritmos de aprendizaje automático para detección de objetos proporcionados en MATLAB[®], se utiliza el algoritmo ACF (*Aggregate Channel Features*) [23]. Este algoritmo corresponde a un detector basado en características, que extrae las características de cada canal en una imagen. Canales comunes puede ser, por ejemplo, el color del píxel, ya sea en RGB o en escala de grises. Existen otros algoritmos para la detección de personas, detección de código de barras, o detectar el fondo en una imagen, que pueden consultarse en la documentación de *Computer Vision Toolbox*. Sin embargo, para este trabajo solo se considera el algoritmo de detección de objetos, ya que es el más adecuado para detectar las balizas.

Con el objetivo de que el rendimiento del algoritmo sea satisfactorio, debe seguirse el siguiente proceso para su entrenamiento y posterior evaluación:



Datos de verdad terreno

Los datos verdad terreno (*Ground Truth Data*) es un término utilizado para referirse a aquella información que se sabe que es correcta. La diferencia de un set de datos verdad terreno respecto a otros sets de datos reside en que los primeros incluyen anotaciones e información adicional. En este caso, las anotaciones corresponden a cuadros delimitadores indicando la localización de la baliza en un fotograma (*frame*) determinado. Como se ha comentado en la sección 6.2, se definen dos misiones distintas para extraer dos sets de datos, uno de entrenamiento y otro de evaluación. Los datos se obtienen guardando en formato de vídeo las imágenes extraídas por el sensor de cámara, explicado en la sección 6.3, haciendo uso de los bloques incluidos en Simulink[®]. Mediante estos vídeos, se hace uso de la aplicación *Video Labeler* proporcionada por MATLAB[®], la cual permite etiquetar los datos de verdad terreno.

En la Figura 6.15 se puede observar un fotograma en el que se identifican las balizas, mediante cuadros delimitadores que indican la región de interés (*ROI - Region Of Interest*). Estas regiones deben ser identificadas en cada fotograma del vídeo, pero la realización manual de este proceso es extenuante. Por este motivo, se utiliza la aplicación *Video Labeler*, ya que dispone de algoritmos integrados para hacer un seguimiento automático de los objetos. De esta forma, basta con identificar el objeto en el primer fotograma que aparece y automatizar el resto, modificando el cuadro delimitador en fotogramas puntuales si hiciera falta. Una vez finalizado el proceso, es posible exportar los datos a un archivo de datos, que posteriormente será utilizado por las funciones de entrenamiento del detector.

El proceso de obtención de datos de verdad terreno debe ser realizado para dos sets de datos, uno de entrenamiento y otro de evaluación. Mediante el set de evaluación se comprobará la precisión del modelo, de forma que se pueda cuantificar su rendimiento.

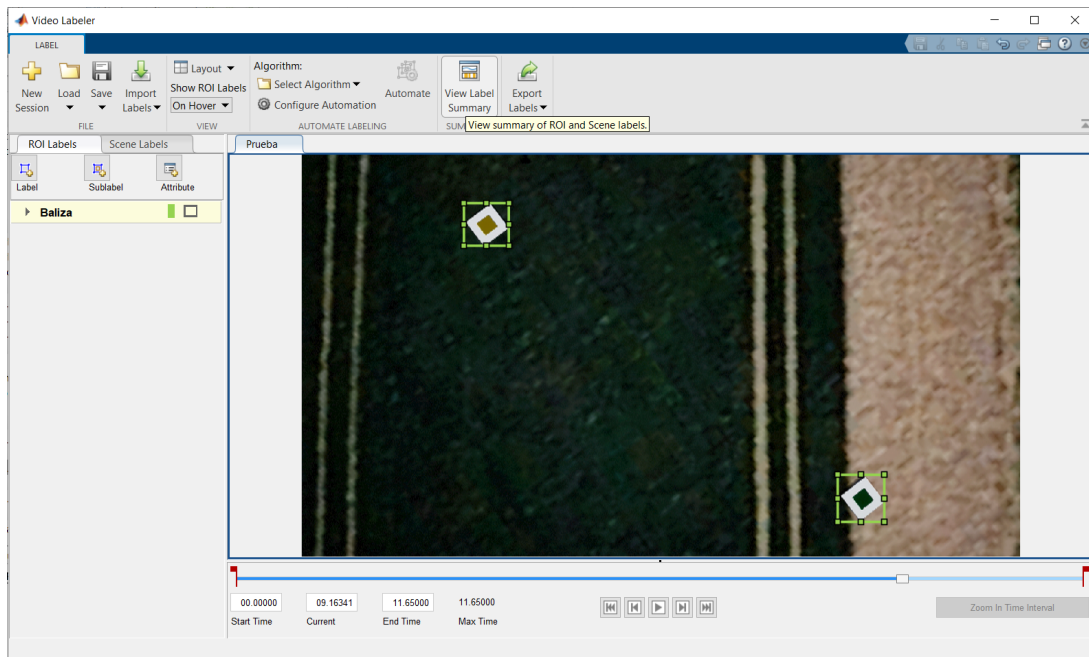


Figura 6.15. Aplicación *Video Labeler*

Entrenamiento del detector

El entrenamiento del detector se realiza mediante la función `trainACFObjectDetector` de MATLAB[®], la cual devuelve el detector entrenado a partir del set de entrenamiento creado. El proceso de creación del set de entrenamiento se explica en detalle en el manual de usuario [22].

Esta función dispone de diversos parámetros que pueden ser configurados por el usuario:

- Factor de muestras negativas (*NegativeSamplesFactor*): indica el número de muestras negativas que se toma para entrenar el modelo. Las muestras negativas son todas las regiones en un fotograma que no corresponden con las determinadas por el cuadro delimitador.
- Número de etapas de entrenamiento (*NumStages*): es el número de veces que se ejecuta el proceso de entrenamiento iterativo. Cuantas más etapas tenga, es más probable que el rendimiento del detector aumente y disminuyan los errores de detección, a costa de un tiempo de entrenamiento mayor.
- Máximo número de clasificadores débiles (*MaxWeakLearners*): los clasificadores débiles son aquellos cuya exactitud es ligeramente superior a la de un clasificador aleatorio. Mediante el algoritmo de aprendizaje automático denominado *Boosting*, se combinan los resultados de varios clasificadores débiles para generar un clasificador robusto, con una exactitud mucho mayor a la de un clasificador aleatorio. Al aumentar el número de clasificadores débiles, es posible aumentar la exactitud en la detección de las balizas, a costa de una velocidad de detección menor.

La elección de estos parámetros está ligada a la aplicación del detector. En primera instancia, se considerarán los valores por defecto en la función. Según el rendimiento obtenido con estos parámetros, se aceptará el detector o se modificará para aumentar su rendimiento.

Evaluación del detector

Con el detector ya entrenado, el siguiente paso es evaluar el rendimiento del mismo. Para ello se utilizan los datos de verdad terreno generados para el set de evaluación. En esta etapa se comparan los resultados obtenidos por el detector con los resultados reales definidos en el set de evaluación.

El rendimiento del detector se evalúa analizando su tasa de fallos y su precisión, utilizando las funciones proporcionadas en MATLAB®, *evaluateDetectionMissRate* y *evaluateDetectionPrecision*, respectivamente. En concreto, se obtienen la precisión, la exhaustividad y la tasa de fallos. Estos parámetros vienen dados por [24]:

$$\text{Precisión} = \frac{VP}{VP + FP} \quad (6.5)$$

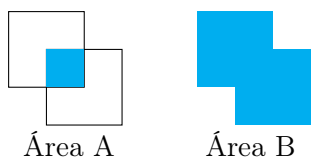
$$\text{Exhaustividad} = \frac{VP}{VP + FN} \quad (6.6)$$

$$\text{Tasa de fallos} = \frac{FN}{VP + FN} \quad (6.7)$$

Donde, $VP = \text{Verdaderos Positivos}$, $FP = \text{Falsos Positivos}$ y $FN = \text{Falsos Negativos}$. La precisión indica la proporción de identificaciones que son correctas, mientras que la exhaustividad (en inglés denominado *recall*) indica la proporción de positivos reales correctamente identificados. Por otro lado, la tasa de fallos indica la proporción de positivos reales no identificados.

En concreto, la precisión y la exhaustividad se utilizan para medir la calidad del modelo entrenado. Un modelo puede tener una alta precisión pero un valor muy bajo de exhaustividad, lo que quiere decir que solo identifica una pequeña parte de las instancias relevantes (en este caso las balizas), pero las instancias detectadas son correctas (Verdaderos Positivos). Por otro lado, se puede tener un modelo que identifique la mayoría de instancias relevantes (valor de exhaustividad elevado), con una precisión baja, indicando que identifica un mayor número de instancias, aumentando la probabilidad de acertar una relevante a costa de un mayor número de falsos positivos.

Un parámetro que debe tenerse en cuenta en las funciones para evaluar el rendimiento del algoritmo es la superposición de los cuadros delimitadores. Para determinar si se ha producido un verdadero positivo o no, la función analiza el porcentaje de superposición entre el cuadro delimitador dado por el detector y el dado por los datos de verdad terreno. Si este porcentaje es superior al umbral establecido en los parámetros de entrada de la función, se considera como un verdadero positivo. Esta relación es calculada como el área de intersección sobre el área de unión (en inglés se denomina *IoU - Intersection over Union*):

$$\text{Superposición} = \frac{\text{Área A} \cap \text{Área B}}{\text{Área A} \cup \text{Área B}}$$


El resultado obtenido tras la evaluación del detector se da en la Figura 6.16, en la cual se establece una superposición del 50%. Se puede comprobar que, un valor alto de exhaustividad produce una disminución en la precisión del algoritmo, como se ha explicado anteriormente. Por otro lado, la gráfica de la tasa de fallos demuestra que, al aumentar el umbral de falsos positivos por imagen (FPPI), se reduce la tasa de fallos. Esto se debe a que, si se considera un valor mayor

de FPPI, los verdaderos positivos tienen mayor probabilidad de ser detectados, disminuyendo así los falsos negativos.

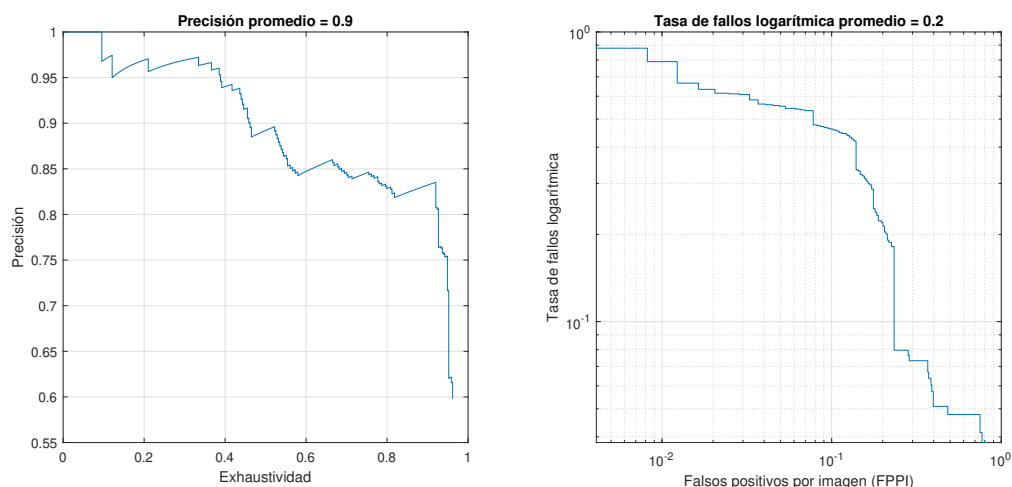


Figura 6.16. Rendimiento del detector con los parámetros por defecto.

Si bien los resultados de la Figura 6.16 permiten visualizar el rendimiento del detector entrenado, no son una buena métrica para comparar entre detectores entrenados con diversos parámetros. Para ello se opta por utilizar la métrica conocida como valor-F1 (en inglés *F1-score*), el cual combina los datos de precisión y exhaustividad en un solo valor, permitiendo una sencilla comparación del rendimiento entre varios detectores [24].

Los resultados obtenidos tras una serie de pruebas, en las que se modificaban los parámetros de entrenamiento del detector, son los dados en la Tabla 6.2. En esta tabla se proporciona el valor medio del valor F1 según los parámetros elegidos para entrenar al modelo. De acuerdo a estos resultados, el primer detector es el que tiene un rendimiento mayor. Para el set de datos utilizado, y según las pruebas realizadas, no es posible conseguir un valor más elevado. Por lo tanto, el detector finalmente utilizado es el detector 1.

	Número de etapas	Factor de muestras negativas	Número de clasificadores débiles	Valor-F1
Detector 1	5	5	2048	0.645
Detector 2	10	5	2048	0.645
Detector 3	5	10	2048	0.469
Detector 4	5	5	3000	0.640
Detector 5	5	2	4000	0.275
Detector 6	5	10	4000	0.470

Tabla 6.2. Comparativa del rendimiento entre los detectores entrenados

6.4.3. Obtención de las coordenadas de las balizas

Mediante la posición de los cuadros limitadores que identifican las balizas, es posible calcular la posición de las mismas a partir de los datos ópticos de la cámara. El sensor de cámara, explicado en la sección 6.3, corresponde a un modelo de cámara estenopeica. Las características de este modelo de cámara se dan en la Figura 6.17.

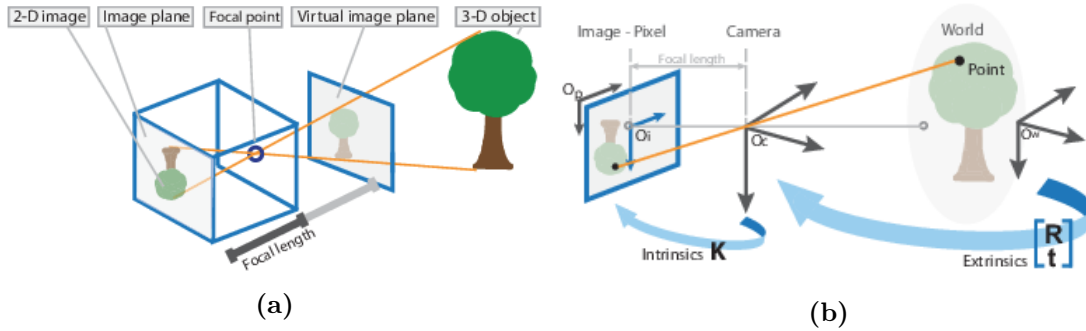


Figura 6.17. Modelo de cámara estenopeica. Recuperado de la documentación de *Computer Vision Toolbox*, de MATLAB[®]

Para poder calcular correctamente la posición de las balizas, se necesitan los parámetros de la cámara utilizada. En este caso los parámetros vienen dados por la Tabla 6.3, donde el centro óptico se especifica en las coordenadas $[c_x, c_y]$ y el tamaño de la imagen corresponde al número de filas por el número de columnas. Los coeficientes de distorsión radial y tangencial se utilizan para simular la distorsión de las lentes en una cámara real.

En la misión real, fuera del entorno de simulación, los parámetros de la cámara tienen que estar definidos para poder aplicar el algoritmo de cálculo de las coordenadas de las balizas. Estos parámetros se pueden obtener mediante la aplicación *Camera Calibrator*, pero para el sistema de simulación se tomarán los valores definidos en el sensor que simula la cámara, *Simulation 3D Camera*, sin considerar la distorsión de las lentes.

Distancia focal (píxeles)	Centro óptico (píxeles)	Tamaño de imagen (píxeles)	Coefficiente de distorsión radial	Coefficiente de distorsión tangencial
f_x	$[c_x, c_y]$	$[w_y, w_x]$	-	-
1109	[640, 360]	[720, 1280]	[0, 0]	[0, 0]

Tabla 6.3. Parámetros de la cámara utilizada en la simulación

El procedimiento para calcular las coordenadas de las balizas es por tanto:

1. Obtener el vector normalizado que indica la dirección en la que se encuentra la baliza en el sistema de coordenadas de la cámara ($[X_c, Y_c, Z_c]$), cuyo origen se sitúa en el centro óptico de la misma (OC). La localización de la baliza en la imagen viene dada por sus coordenadas en el sistema de referencia de píxeles (x, y) , por lo que a partir de estos datos y, conociendo los parámetros de la cámara, se puede obtener el vector que indica la dirección de la baliza, según la ecuación 6.8, el cual debe ser normalizado. La implementación de este cálculo se indica en la línea 26 y 27 de la Lista 6.1.

Para facilitar la comprensión del problema en 3 dimensiones, se proporciona la representación

de la Figura 6.18, donde la imagen visualizada se representa en color gris.

$$\vec{A}_c = [x - c_x, y - c_y, f_x]^T \quad (6.8)$$

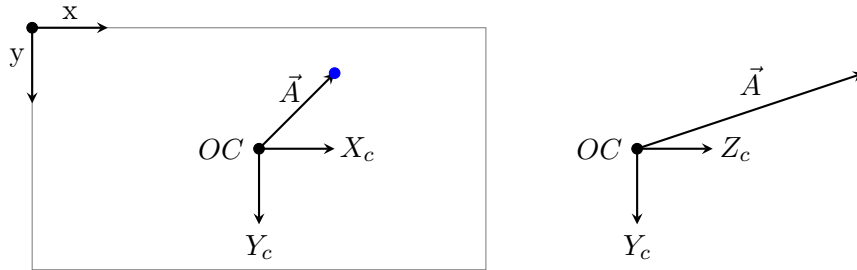


Figura 6.18. Vector normalizado indicando la baliza

2. Transformar el vector del sistema de coordenadas de la cámara a ejes cuerpo: en este caso basta con multiplicar el vector anterior por la matriz de rotación. Para ello hay que tener en cuenta la orientación de la cámara, definida en sus parámetros. En este caso su eje Z_c coincide con el eje Z_b del sistema de coordenadas ejes cuerpo, por lo que la matriz de rotación corresponde a una rotación de 90° sobre el eje Z . Con esto, el vector \vec{A} en ejes cuerpo viene dado por la expresión 6.9, cuya implementación se da en la línea 28 de la Lista 6.1.

$$\vec{A}_b = \begin{pmatrix} \cos(-\pi/2) & \sin(-\pi/2) & 0 \\ -\sin(-\pi/2) & \cos(-\pi/2) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \vec{A}_c \quad (6.9)$$

3. Transformar el vector al sistema coordenadas NED (\vec{A}_e): para ello se utiliza la matriz de rotación de ejes cuerpo a ejes tierra, que es igual a la traspuesta de la matriz de rotación de ejes tierra a ejes cuerpo, obtenida en el bloque *6 DoF Equations of Motion* de la sección 6.1. La implementación de este cálculo se da en la línea 29 de la Lista 6.1.
4. Escalar el vector: finalmente, una vez expresado el vector en ejes tierra (NED), se multiplica por una constante k en un bucle. Esto se realiza ya que se desconoce la distancia de la cámara a la baliza, por lo que se procede de la siguiente manera:

- Multiplicar el vector normalizado por un valor k , incrementado con cada iteración del bucle. Al vector resultante se añade la posición de la cámara en el sistema NED, calculado a partir de la posición NED del centro de gravedad de la aeronave y teniendo en cuenta la traslación de la cámara (según los parámetros del bloque *Simulation 3D Camera*). Este paso se muestra en la línea 34 de la Lista 6.1.
- Calcular la altura del terreno en las coordenadas x e y del punto indicado por el vector, cuyo origen se sitúa en la cámara. Los datos de altura del terreno se almacenan en una matriz cuadrada de orden 2017, por lo que a partir de las coordenadas x e y del punto se calcula el elemento de dicha matriz.

Para determinar dicho elemento debe tenerse en cuenta que el elemento (1, 1) de la matriz MDT se sitúa en las coordenadas (1007, -1007) del sistema de referencia NED (noroeste del mapa). Esto es, la fila de la matriz depende de la coordenada x del punto, mientras que la columna depende de la coordenada y del punto, como se muestra en las líneas 35 y 36 de la Lista 6.1.


```

23     for m = 1:length(balizas(balizas(:,1) ~= 0))
24         xBaliza = min(balizas(m,1) + balizas(m,3)/2, wx);
25         yBaliza = min(balizas(m,2) + balizas(m,4)/2, wy);
26         Ac = [xBaliza - ocx; yBaliza - ocy; f];
27         Ac = Ac / norm(Ac);           % normalizar el vector
28         Ab = [-Ac(2); Ac(1); Ac(3)]; % transformar a ejes cuerpo
29         Ae = DCMeB * Ab;           % transformar a ejes tierra
30         posCamara = NED + DCMeB*traslacionCamara';
31
32         % Localizar la baliza en tierra
33         for k = 1:iteracionesBucle
34             Punto = posCamara + single(Pe * k);
35             columnaMDT = round(max(1008 + Punto(2), 1));
36             filaMDT = round(max(1008 - Punto(1), 1));
37             heightDiff = MDT(filaMDT, columnaMDT) + Punto(3);
38             datosMDT(k, :) = [Punto(1), Punto(2), Punto(3), abs(heightDiff)];
39         end
40         [~, pos] = min(datosMDT(:,4));
41         NED_P(m,:) = [datosMDT(pos,1), datosMDT(pos,2), datosMDT(pos,3)];

```

Lista 6.1. Cálculo de las coordenadas de las balizas

- Calcular la diferencia de alturas entre la coordenada z del punto y la elevación del terreno. Este paso se implementa en la línea 37 de la Lista 6.1. Cabe destacar que la coordenada z del punto es positiva si se sitúa por debajo del origen de coordenadas, mientras que la altura del terreno dada por la matriz MDT es positiva si se sitúa por encima del origen de coordenadas.
- Una vez finalizado el bucle, determinar la diferencia de altura mínima y las coordenadas x e y en las que se da. Estas coordenadas serán finalmente la localización de la baliza, como puede observarse en las líneas 40 y 41 de la Lista 6.1.

6.4.4. Implementación en *Simulink*

La implementación final del subsistema de visión artificial puede observarse en la Figura 6.19. El algoritmo de detección de las balizas se implementa en la función *Detectar objetos*, cuya única entrada corresponde con la imagen proporcionada por la cámara. Por otro lado, la obtención de las coordenadas de las balizas se implementa en la función *Calcular coordenadas*, que acepta como entradas los datos de las balizas (cuadros delimitadores dentro de la imagen), la matriz de rotación de ejes cuerpo a ejes tierra, utilizada para el paso 3 del procedimiento, y las coordenadas NED de la aeronave, para determinar el origen de coordenadas del vector. La Lista 6.1 corresponde con parte del código implementado en la función *Calcular coordenadas*.

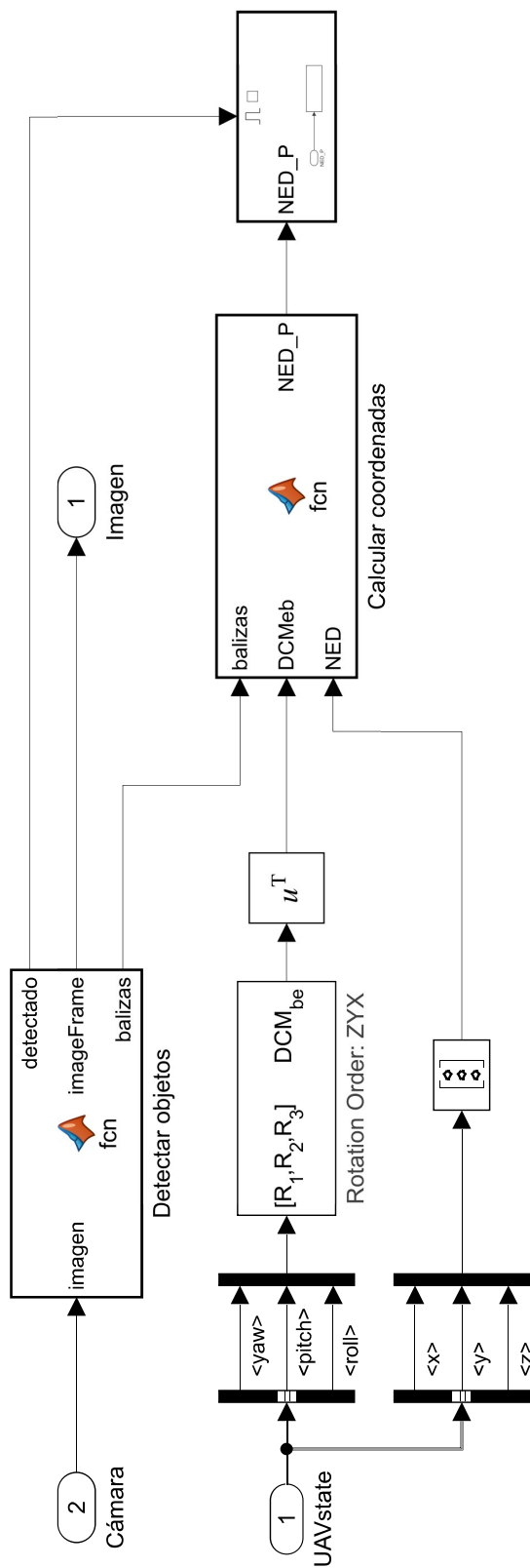


Figura 6.19. Implementación en *Simulink* de la detección de balizas

7 | Resultados

7.1. Misión

La misión utilizada para verificar el sistema de simulación desarrollado se muestra en la Figura 7.1. Corresponde a la misión definida en *QGroundControl*, en la que puede observarse la posición de la aeronave en un instante dado. Esta misión se define de forma que todas las balizas sean visibles desde la aeronave, con el objetivo de obtener el error en el cálculo de las coordenadas de las mismas.



Figura 7.1. Visualización de la misión en QGC.

Las coordenadas LLA, las coordenadas NED y la velocidad elegida en cada waypoint se detalla en la Tabla 7.1. Por otro lado, en la Figura 7.2 se proporciona la representación tridimensional de la trayectoria seguida por la aeronave, junto con los waypoints identificados por triángulos.

Waypoint	1	2	3	4
Latitud (°)	52.8335	52.8399	52.8387	52.8319
Longitud (°)	-0.7755	-0.7740	-0.7634	-0.7642
Altitud (m)	195.1	205.1	200.1	190.1
x (m)	62.9518	778.2536	648.3531	-109.8243
y (m)	21.5386	123.4370	838.1916	786.2622
z (m)	-50	-60	-55	-45
Velocidad (m/s)	20	25	22	20

Tabla 7.1. Waypoints de la misión de simulación

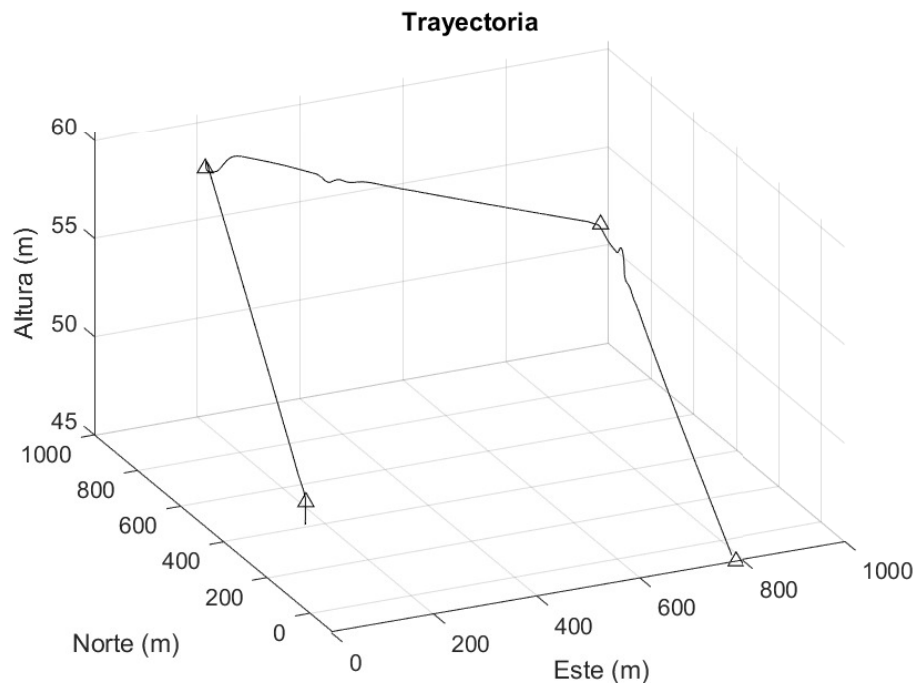


Figura 7.2. Representación de la trayectoria

7.2. Sistema de control

Para verificar la respuesta del sistema de control conviene obtener la evolución temporal de las variables de estado y de las variables de control a lo largo de la trayectoria.

Seguimiento de la trayectoria

Primeramente, en la Figura 7.3 se puede comprobar la evolución de la posición de la aeronave en coordenadas NED. En la misma se observa que la diferencia entre la posición real de la aeronave y la referencia es mayor tras alcanzar cada waypoint, ya que se produce un cambio significativo en la trayectoria. No obstante, al cabo de unos diez segundos esta diferencia disminuye hasta

valores razonables. Por lo tanto, se concluye que el sistema de control es adecuado para seguir la trayectoria definida en la misión.

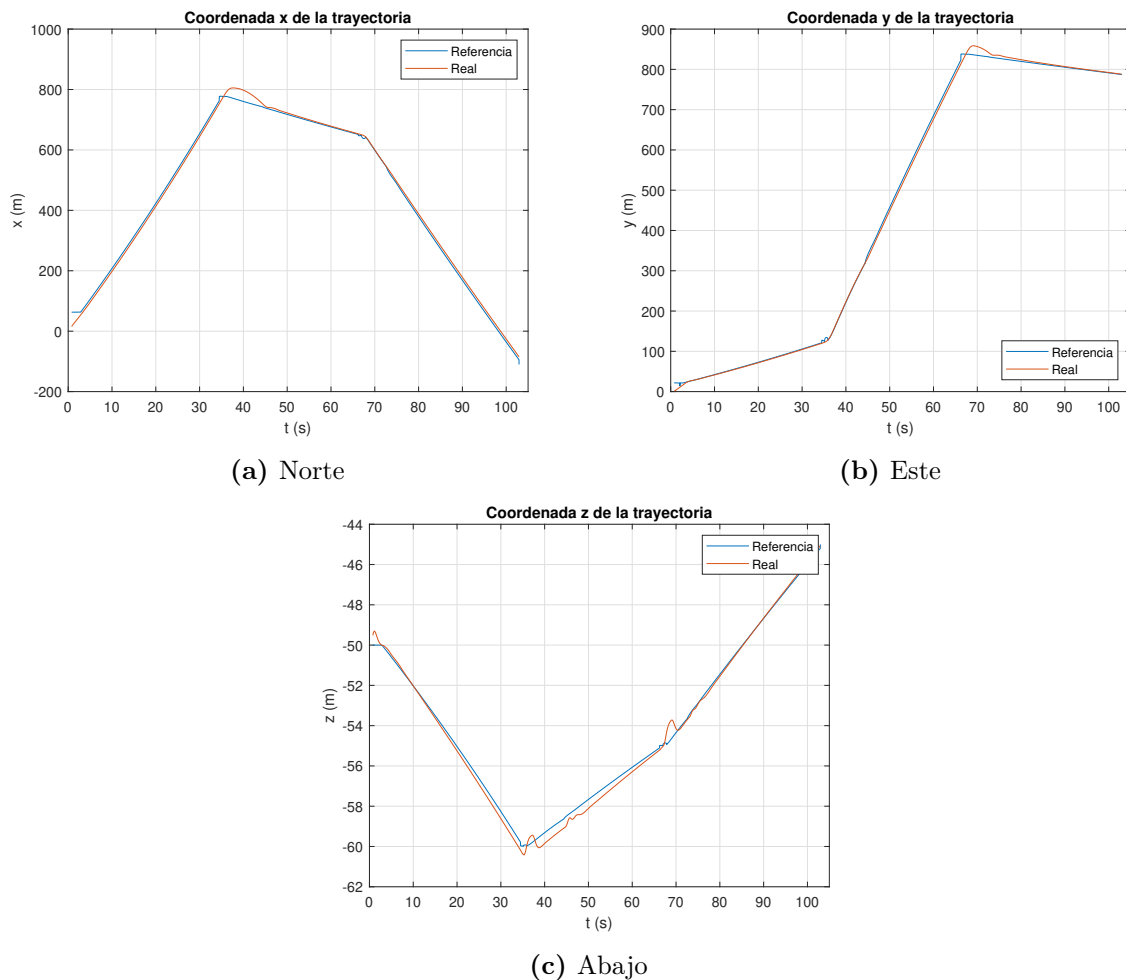


Figura 7.3. Evolución temporal de la posición de la aeronave respecto a la trayectoria de referencia

Acciones de control

Por otro lado, la evolución temporal de las acciones de control, correspondientes al lazo de control interno, es la dada en la Figura 7.4. Comparando estos resultados con la trayectoria seguida por la aeronave, dada en la Figura 7.3, se comprueba que los mayores cambios en las acciones de control se producen tras alcanzar un waypoint, como es de esperar. En estos puntos se producen oscilaciones en la deflexión de las superficies debido a las características del sistema de control utilizado.

Estas oscilaciones deberían reducirse modificando las ganancias de los controladores utilizados, ya que el sistema de control está diseñado para operar en torno a una altura de 150 m, mientras que en esta simulación la aeronave vuela a una altura inferior a 60 m, con el fin de cumplir con las limitaciones del concurso *UAS Challenge*, al igual que obtener una mayor resolución de las balizas en la imagen captada por la cámara. Sin embargo, el ajuste de los controladores está fuera del alcance de este proyecto.

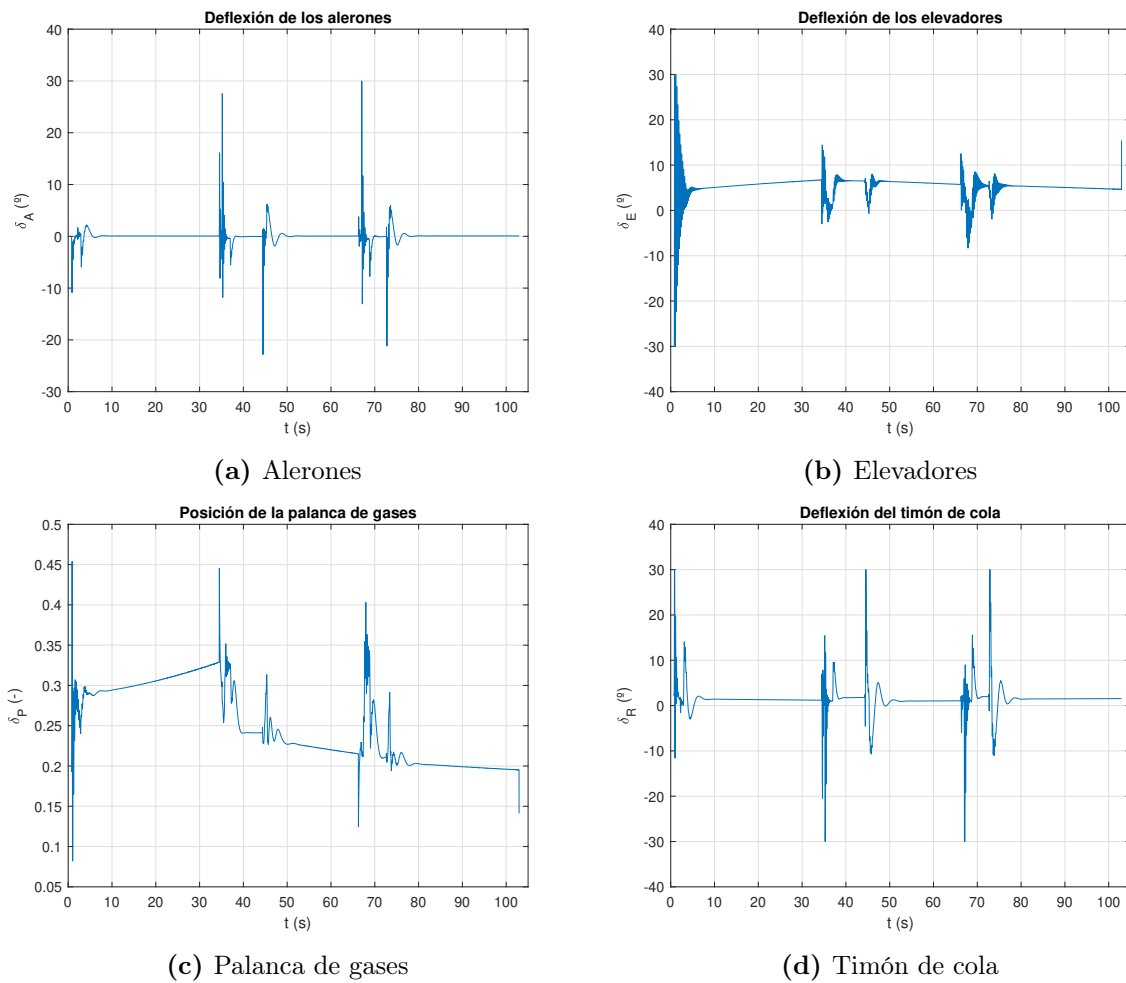


Figura 7.4. Evolución temporal de las acciones de control

Lazo de control externo

El lazo de control externo genera las referencias utilizadas por el lazo de control interno. La evolución temporal de estas referencias se muestra en la Figura 7.5. Según los resultados obtenidos, puede comprobarse que la aeronave sigue adecuadamente las referencias del ángulo de alabeo y cabeceo. Sin embargo, el ángulo de cabeceo real de la aeronave presenta mayores oscilaciones que el ángulo de alabeo, debido a la deflexión de los elevadores, dada en la Figura 7.4b. Como se ha explicado, estas oscilaciones podrían reducirse mediante un reajuste de las ganancias de los controladores. Por otro lado, la aeronave sigue los valores de referencia de la velocidad longitudinal y la aceleración lateral en ejes cuerpo con una precisión alta.

Variables de estado

En cuanto al resto de variables de estado, la representación de su evolución temporal se da en las Figuras 7.6 - 7.9. En la Figura 7.6 se representa la velocidad lateral y vertical en ejes cuerpo, ya que la velocidad longitudinal se da en la Figura 7.5c. En la Figura 7.7 se muestra la velocidad angular en los tres ejes cuerpo de la aeronave.

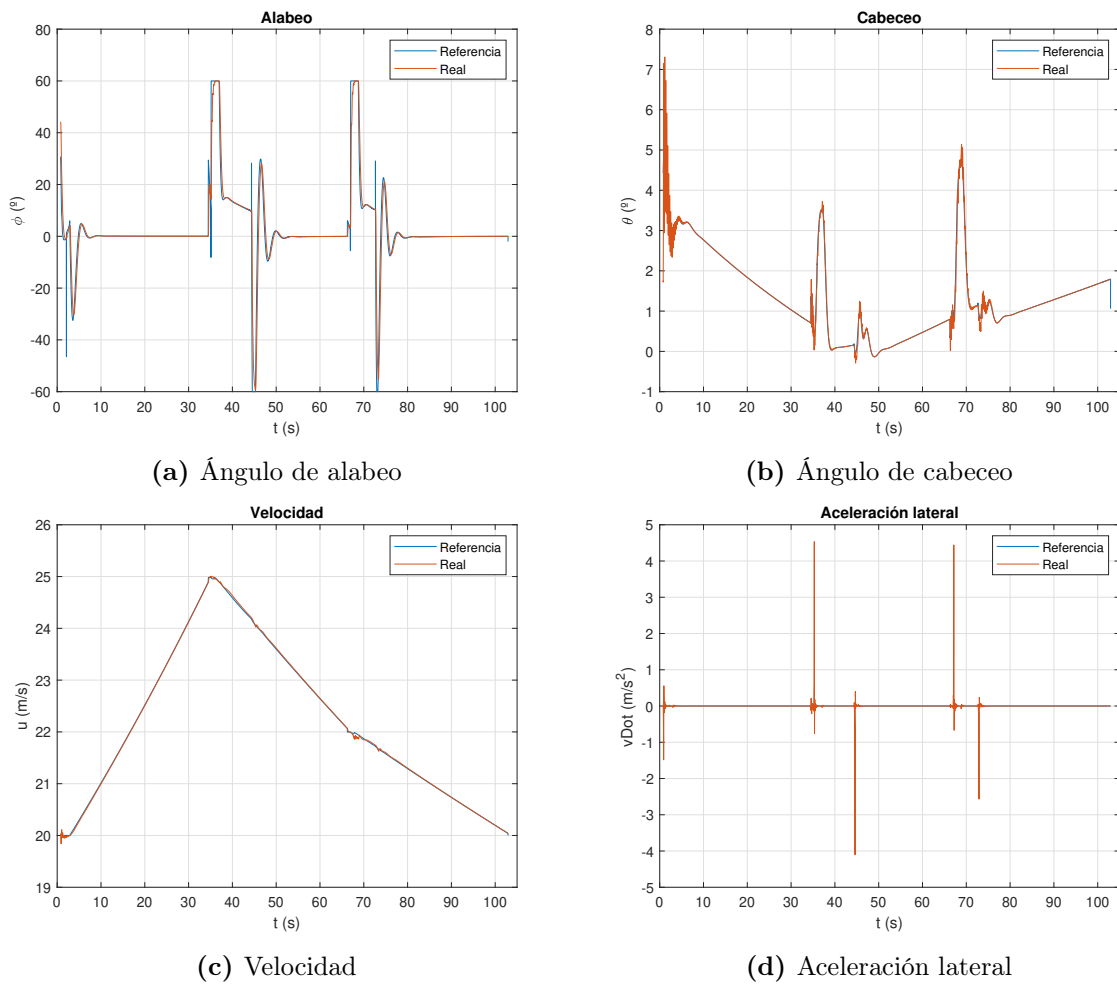


Figura 7.5. Evolución temporal de las referencias emitidas por el lazo de control externo

Por otro lado, en la Figura 7.8 se representa la evolución del ángulo de ataque y derrape, necesarios para el cálculo de las fuerzas y momento aerodinámicos. Puede comprobarse que el ángulo de ataque es menor de 15° , por lo que la aeronave no se encuentra en la zona de entrada en pérdida, indicando que la misión se encuentra dentro de su dominio de vuelo.

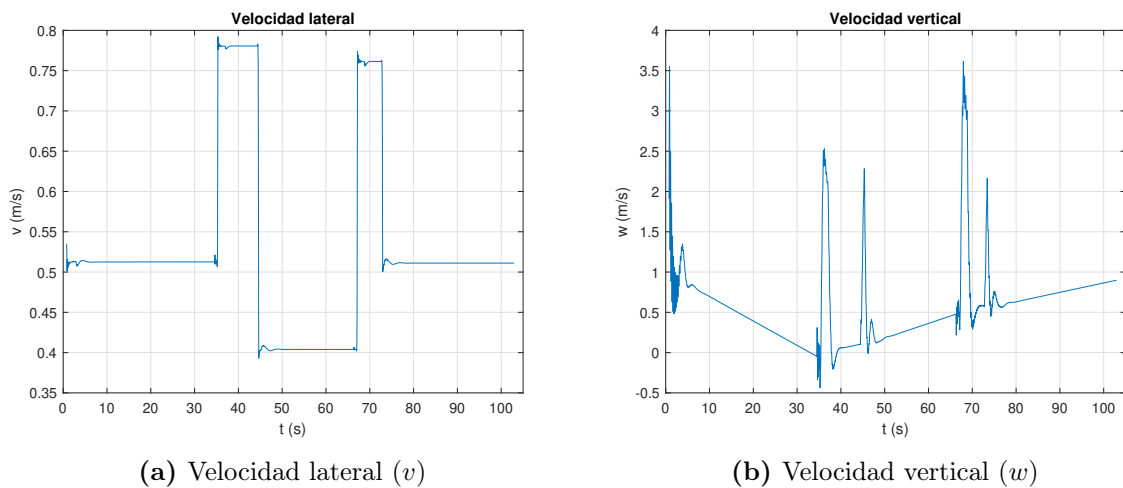


Figura 7.6. Evolución temporal de las velocidades en ejes cuerpo

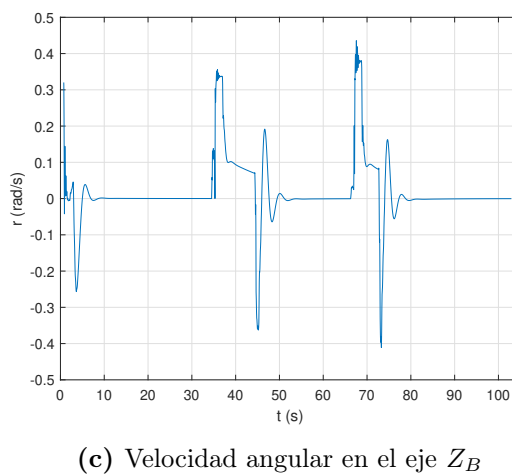
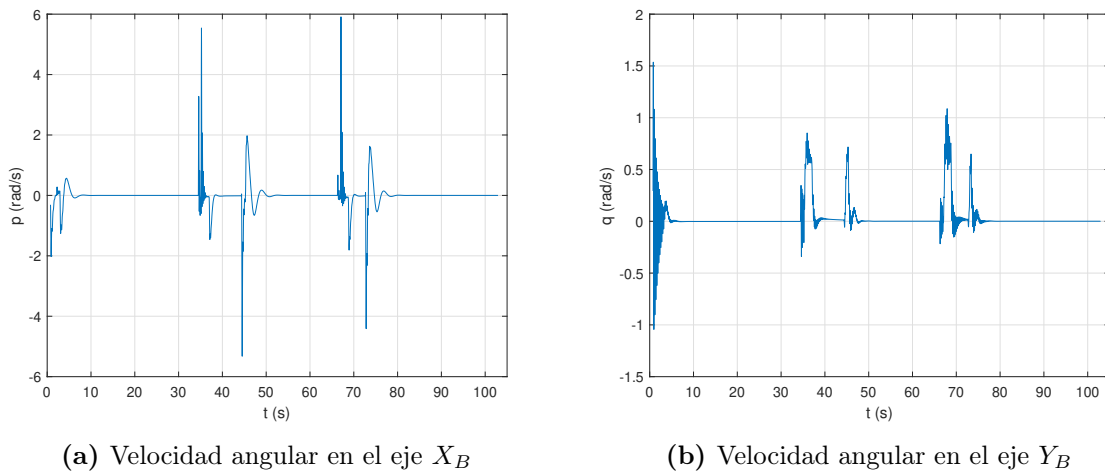


Figura 7.7. Evolución temporal de las velocidades angulares

Finalmente, en la Figura 7.9 se muestra la velocidad de la aeronave en ejes tierra. Cabe destacar que una velocidad en el eje Z_E positiva indica que la aeronave está descendiendo, lo cual ocurre a partir de los 40 s de simulación, tras alcanzar el segundo waypoint, como puede comprobarse en la Figura 7.2.

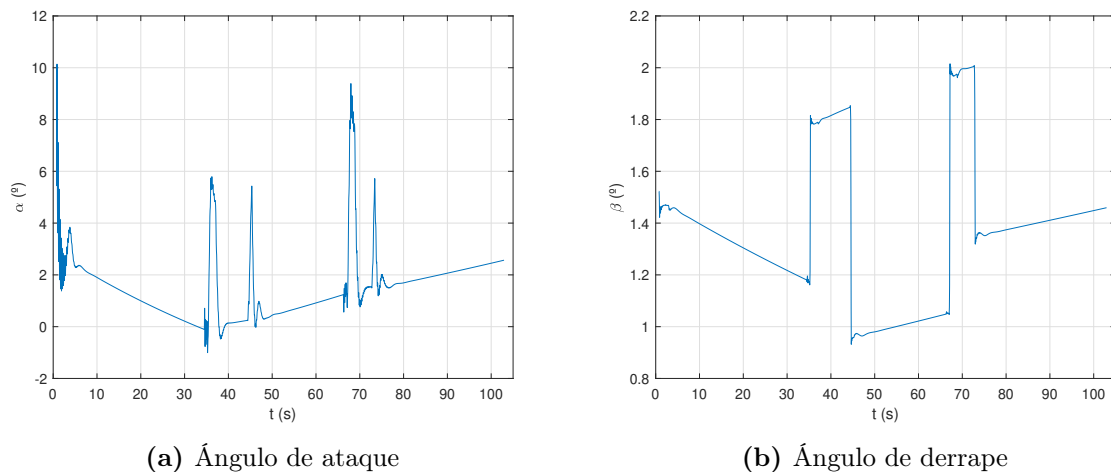


Figura 7.8. Evolución temporal del ángulo de ataque y derrape

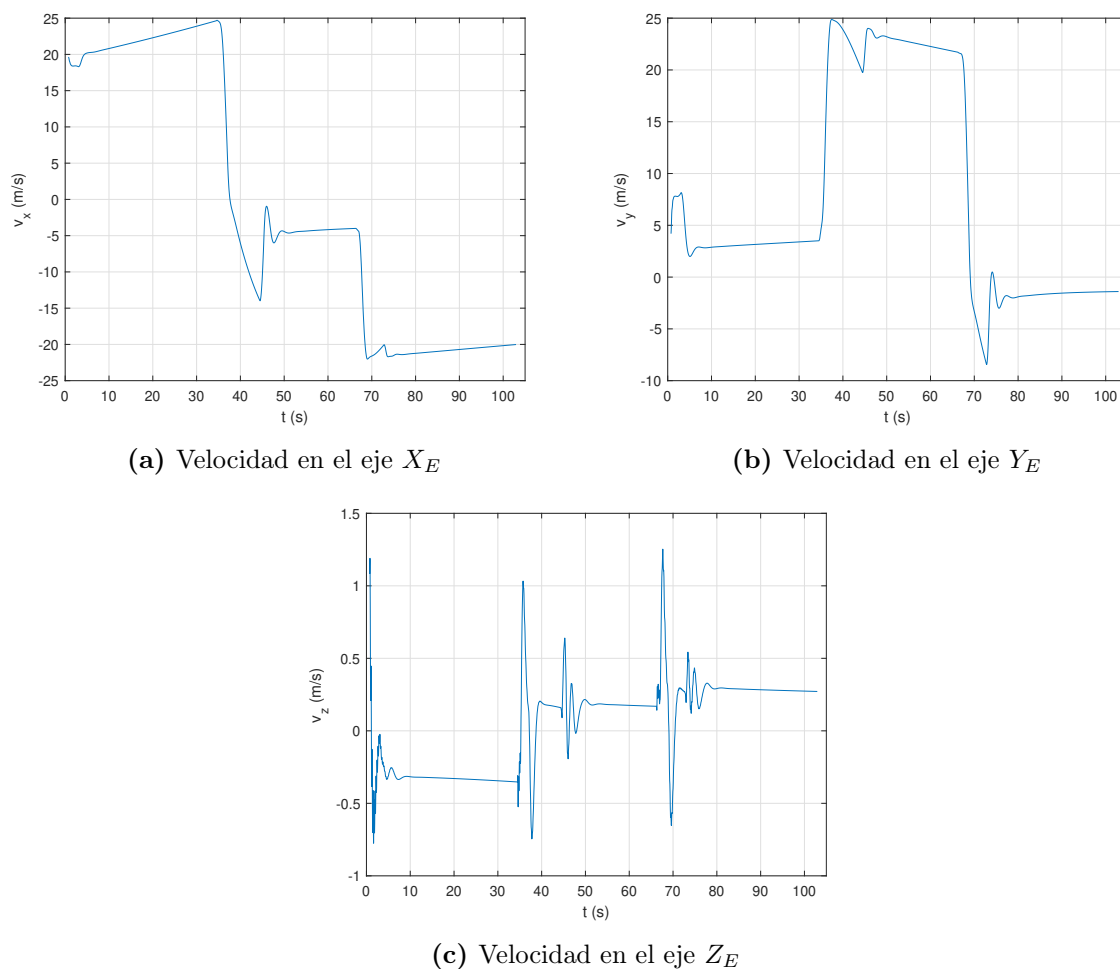


Figura 7.9. Evolución temporal de las velocidades en el sistema de referencia NED

7.3. Visión artificial

La visualización de la aeronave en un instante dado en la misión se muestra en la Figura 7.10, donde se pueden observar las balizas colocadas sobre el terreno. Las imágenes captadas por la cámara unida a la aeronave se muestran en la Figura 7.11, junto con el cuadro delimitador dado por el algoritmo de visión artificial y la confianza de la detección.

Tras la simulación se obtienen las coordenadas de las balizas en el sistema de referencia ejes tierra (NED). Estas coordenadas pueden ser convertidas a coordenadas geodésicas mediante funciones proporcionadas por MATLAB[®]. Sin embargo, para calcular el error en el posicionamiento se trabaja con coordenadas NED, de modo que los errores estén en metros y sea más sencillo comprender su magnitud.

Para calcular el error en la determinación de la posición de cada baliza es necesario disponer de los datos reales de cada baliza, que pueden obtenerse en el editor de Unreal Engine. El algoritmo de visión artificial identifica cada baliza durante la simulación, obteniendo así sus coordenadas NED. Con esto, para cada una de las balizas identificadas debe calcularse la diferencia de posición entre esta y todas las balizas cuyos datos reales son conocidos. Este procedimiento genera un vector con tantos elementos como datos de balizas reales se disponga, en el cual el elemento menor indica de qué baliza se trata. En resumen, el procedimiento que se sigue es el siguiente:

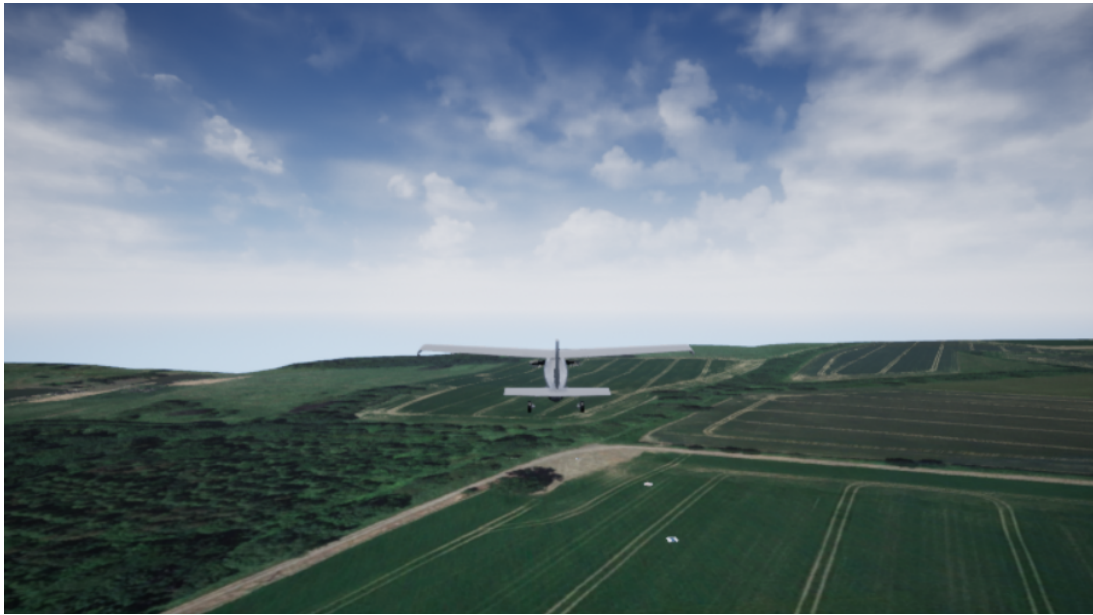
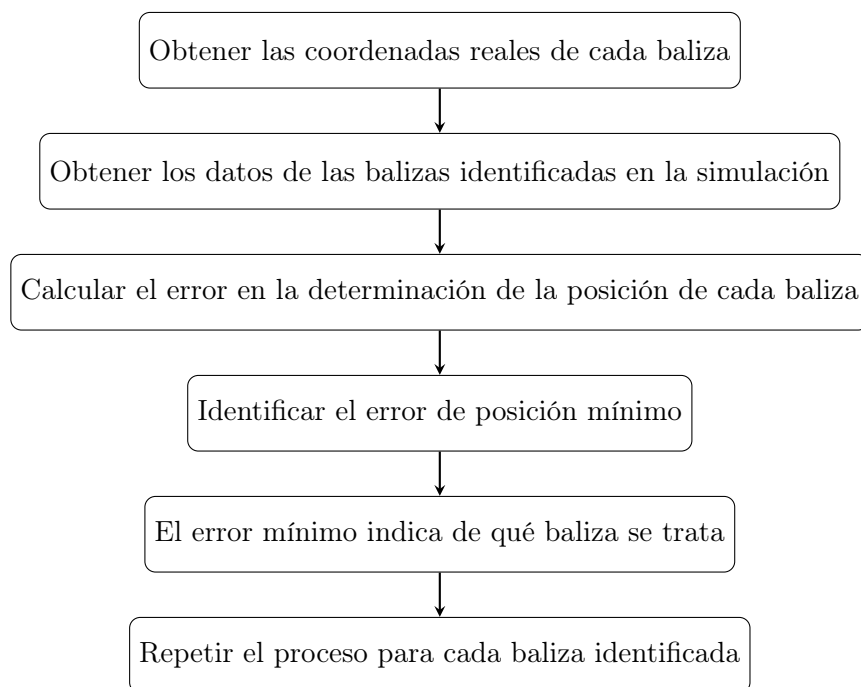


Figura 7.10. Visualización de la aeronave en el escenario realista



Los errores obtenidos en cada coordenada (x, y, z) se muestran en la Figura 7.12. Según estos resultados, se comprueba que tanto el algoritmo de visión artificial, como el cálculo de las coordenadas a partir de la posición de la baliza en la imagen, funcionan correctamente, demostrando así su validez.

La principal fuente del error en el cálculo de la posición viene dado por la diferencia del punto central del cuadro delimitador y el punto central de la baliza. Teniendo en cuenta que la posición de la baliza viene determinada por la dirección del vector calculado (que indica el centro del cuadro delimitador), si este se desvía una pequeña cantidad respecto al vector real (que indica el centro de la baliza en la imagen), la diferencia en la posición del punto final de ambos vectores se vuelve significativa a una altura de 50 m. Cuanto menor sea la altura de la aeronave, menor



Figura 7.11. Visualización de la imagen captada por la cámara, con la identificación de la baliza

será el error, y viceversa. Así mismo, otra fuente de error corresponde con la resolución del mapa creado en Unreal Engine, y la fidelidad con la que replica el terreno real. Por último, el algoritmo de cálculo de la posición introduce errores, ya que se trata de obtener el valor más cercano, pero no exacto. Sin embargo, estas dos últimas fuentes de errores son irrisorias en comparación con la primera.

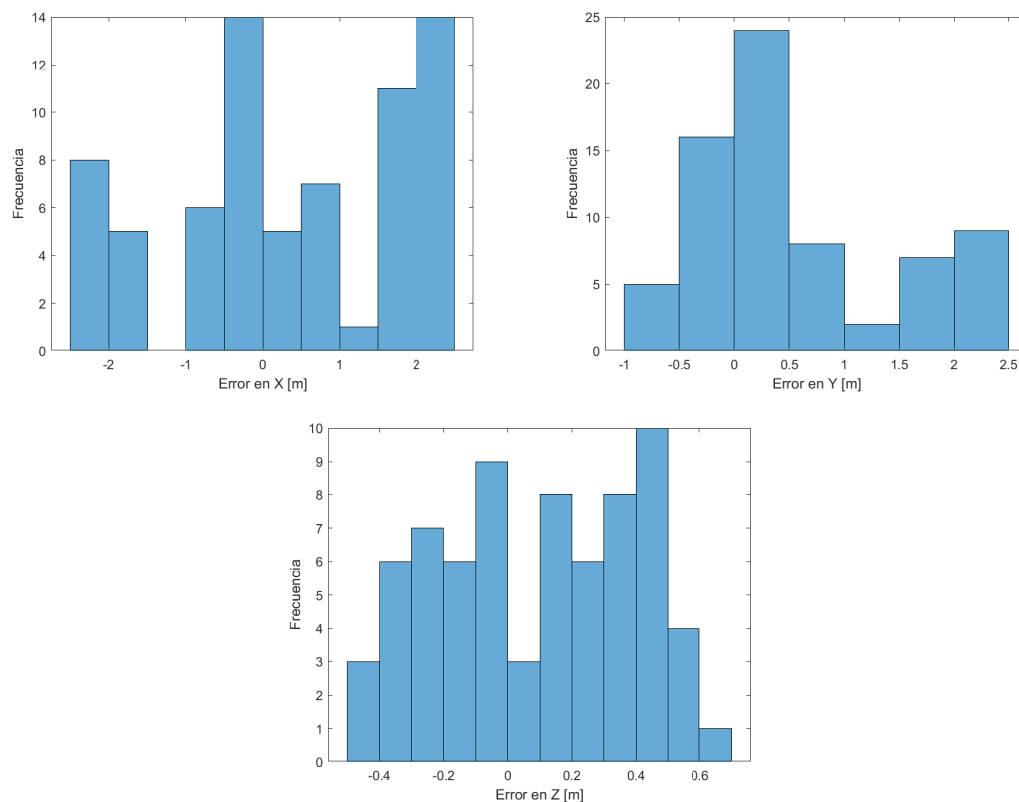


Figura 7.12. Errores en la determinación de la posición de las balizas

8 | Conclusiones y trabajos futuros

La realización de este trabajo ha permitido integrar en un mismo sistema de simulación el modelo dinámico que describe el comportamiento de una aeronave, el sistema de control y guiado que permite seguir una ruta definida, un entorno de visualización realista del estado de la aeronave, y algoritmos de visión artificial que analizan la imagen recuperada del entorno para identificar objetos dentro de este. Mediante visión artificial se ha conseguido la identificación de las coordenadas de balizas terrestres incluidas en el entorno realista, con una precisión horizontal menor a 2.5 m y una precisión vertical menor a 0.7 m.

El producto de este trabajo servirá como una plataforma de simulación para misiones reales de sistemas aéreos no tripulados, en la que se podrá verificar las características de los algoritmos de guiado y control diseñados, al igual que el rendimiento de algoritmos de visión artificial. Posteriormente, la validación de los algoritmos en esta plataforma de simulación podrá dar paso a ensayos no destructivos mediante técnicas de hardware-in-the-loop, siendo estos el paso previo a la implementación en la aeronave real y su verificación en ensayos de vuelo.

Por lo que se refiere a las limitaciones de este trabajo, el entorno de visualización realista, si bien se desarrolla en las cercanías de la zona del concurso *UAS Challenge*, no corresponde a la zona exacta del concurso. Paralelamente, la misión simulada no incorpora las fases de vuelo de despegue y aterrizaje. Del mismo modo, los algoritmos utilizados no han sido optimizados, por lo que la respuesta obtenida dista de ser ideal.

En definitiva, estas limitaciones deberán ser solventadas mediante futuros trabajos, haciendo hincapié en ampliar el sistema de simulación para permitir el desarrollo de la misión completa, desde la puesta en marcha de la aeronave y su despegue, hasta el aterrizaje de la misma. Además, las características del modelo dinámico deberán ser modificadas según se produzcan cambios en la geometría y el diseño de la aeronave. Del mismo modo, será necesario reajustar el sistema de control con el fin de mejorar el seguimiento de la ruta definida en las misiones, al igual que incorporar algoritmos para la detección de los caracteres en las balizas y el color de las mismas.

Bibliografía

- [1] Luis Llamas. Teoría de control en Arduino: El controlador PID. Disponible en: <https://www.luisllamas.es/teoria-de-control-en-arduino-el-controlador-pid/>, Fecha de consulta: julio de 2021.
- [2] Gerardo Acosta. Taller de controlador basado en lógica difusa. *Universidad Nacional del Centro de la Provincia de Buenos Aires. Publicación Independiente, Noviembre 2001*, 1996.
- [3] Elizabeth Villota Cerna. Problema del Regulador Cuadrático Lineal (LQR, por sus siglas en inglés), 2010. Clase 10-02, Semestre 2010I-UNI. CONTROL MODERNO Y ÓPTIMO, Universidad Nacional de Ingeniería - Facultad de Ingeniería Mecánica.
- [4] Diego Fernando Sendoya. ¿ qué es el control predictivo y hacia dónde se proyecta? *Publicaciones e Investigación*, 7:53–59, 2013.
- [5] Sergio Torrubia Caravaca. *Redes neuronales multimodelo aplicadas al control de sistemas*. Proyecto de Ingeniería Técnica en Informática de Sistemas, Universitat Autònoma de Barcelona, 2010.
- [6] The MathWorks Inc. UAV Toolbox. Disponible en <https://es.mathworks.com/help/uav/>, Fecha de consulta: abril de 2021.
- [7] Cristina Cuerno-Rejado, Luis García-Hernández, Alejandro Sánchez-Carmona, Adrián Carrío, José Luis Sanchez-Lopez, and Pascual Campoy. Evolución histórica de los vehículos aéreos no tripulados hasta la actualidad. *Dyna*, 91(3):282–288, 2016.
- [8] Epic Games Inc. Unreal Engine 4 Documentation - Landscape Technical Guide. Disponible en: <https://docs.unrealengine.com/4.26/en-US/BuildingWorlds/Landscape/TechnicalGuide/>, Fecha de consulta: junio de 2021.
- [9] OACI. Circular 328, *Sistemas de aeronaves no tripuladas (UAS)*. Technical report, Organización de Aviación Civil Internacional.
- [10] Institution of Mechanical Engineers. Challenge document library. Disponible en <https://www.imeche.org/events/challenges/uas-challenge/team-resources/challenge-document-library>, Fecha de consulta: mayo de 2021.
- [11] Haiyang Chao, Yongcan Cao, and YangQuan Chen. Autopilots for small fixed-wing unmanned air vehicles: A survey. In *2007 International Conference on Mechatronics and Automation*, pages 3144–3149, 2007.
- [12] Daniel Villalibre Vilariño. *Diseño del Sistema de Control y Navegación para una Aeronave No Tripulada dentro del Proyecto HERMES-UPV*. Trabajo Fin de Grado, Universitat Politècnica de València, 2019.

-
- [13] Priya Dialani. Top 10 Computer Vision Tools for 2020. Disponible en: <https://www.analyticsinsight.net/top-10-computer-vision-tools-for-2020/>, Fecha de consulta: junio de 2021.
- [14] Álvaro Goterris Fuster. *Desarrollo del Modelo Dinámico No Lineal de una aeronave no tripulada para el proyecto HERMES-UPV*. Trabajo Fin de Grado, Universitat Politècnica de València, 2019.
- [15] Sanghyuk Park, John Deyst, and Jonathan How. A new nonlinear guidance logic for trajectory tracking. In *AIAA guidance, navigation, and control conference and exhibit*, page 4900, 2004.
- [16] The MathWorks Inc. Path Manager. Disponible en https://es.mathworks.com/help/uav/ref/pathmanager.html?searchHighlight=path%20manager&s_tid=srchtitle, Fecha de consulta: abril de 2021.
- [17] The MathWorks Inc. Waypoint Follower. Disponible en https://es.mathworks.com/help/uav/ref/waypointfollower.html?searchHighlight=waypoint%20follower&s_tid=srchtitle, Fecha de consulta: abril de 2021.
- [18] Christian Greuel. Real-world Terrain Visualization with Unreal Engine. Disponible en <https://www.linkedin.com/pulse/real-world-terrain-visualization-unreal-engine-christian-greuel>, Fecha de consulta: junio de 2021.
- [19] Instituto Nacional de Estadística Geografía e Informática. Modelos Digitales de Elevación (MDE). Disponible en <https://www.inegi.org.mx/contenidos/temas/mapas/relieve/continental/metadatos/mde.pdf>, Fecha de consulta: junio de 2021.
- [20] GISGeography. 5 Free Global DEM Data Sources - Digital Elevation Models. Disponible en <https://gisgeography.com/free-global-dem-data-sources/>, Fecha de consulta: junio de 2021.
- [21] Department for Environment Food & Rural Affairs. Defra Survey Data Download. Disponible en <https://environment.data.gov.uk/DefraDataDownload/?Mode=survey>, Fecha de consulta: junio de 2021.
- [22] Arnau García Cases. *Diseño de un sistema de simulación de misiones autónomas para sistemas aéreos no tripulados. Aplicación al proyecto HORUS UPV - Manual de usuario*, 2021.
- [23] The MathWorks Inc. Aggregate Channel Features (ACF) object detector. Disponible en https://es.mathworks.com/help/vision/ref/acfobjectdetector.html?searchHighlight=ACF&s_tid=srchtitle, Fecha de consulta: junio de 2021.
- [24] Jose Martinez Heras. Precision, Recall, F1, Accuracy en clasificación. Disponible en <https://www.iartificial.net/precision-recall-f1-accuracy-en-clasificacion/>, Fecha de consulta: junio de 2021.



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



TRABAJO FIN DE GRADO

Diseño de un sistema de simulación de misiones
autónomas para sistemas aéreos no tripulados.
Aplicación al proyecto HORUS UPV

Pliego de condiciones

AUTOR

GARCÍA CASES, Arnau argarca@etsid.upv.es

TUTOR

GARCÍA-NIETO RODRÍGUEZ, Sergio sgnieto@isa.upv.es

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO

GRADO EN INGENIERÍA AEROESPACIAL
CURSO 2020-2021

Índice

1	Introducción	1
2	Descripción del proyecto	1
3	Pliego de condiciones generales	1
3.1	Documentación del proyecto	1
3.2	Condiciones Generales Facultativas	2
3.3	Condiciones de la Ejecución del Proyecto	2
3.4	Condiciones Generales Económicas	3
4	Pliego de prescripciones técnicas	4
4.1	Objeto	4
4.2	Condiciones de los materiales	4
4.3	Condiciones de ejecución	5
4.4	Pruebas y ajustes finales o de servicio	6

1. Introducción

El presente documento contiene el pliego de condiciones del proyecto “Diseño de un sistema de simulación de misiones autónomas para sistemas aéreos no tripulados. Aplicación al proyecto HORUS UPV”, en el cual se detallan las condiciones técnicas y legales necesarias para el desarrollo del mismo. Asimismo, señala las responsabilidades, derechos y obligaciones entre contratante y contratista para la ejecución del proyecto.

2. Descripción del proyecto

Las partes en las que se desarrolla este proyecto son:

1. Modificación del modelo no lineal en MATLAB[®] y Simulink[®] para adecuarse a la estructura del sistema de simulación diseñado. La descripción matemática de la aeronave será proporcionada por el contratante, que incluye las características geométricas, aerodinámicas y el modelo propulsivo, así como el sistema de control de la misma.
2. Generación de un entorno de simulación que sirva para visualizar el estado de la aeronave durante la misión.
3. Generación de balizas en el entorno de simulación, cuya posición y características podrán ser modificadas acorde a las necesidades del contratante.
4. Establecimiento de una conexión del entorno de simulación con estaciones de control terrestres, externas al mismo. El contratante podrá modificar la misión mediante esta estación.
5. Implementación de algoritmos de visión artificial para permitir la capacidad de identificación de objetos en el entorno de simulación.
6. Cálculo de las coordenadas de las balizas a partir de su identificación durante la simulación de la misión.
7. Validación del sistema mediante la comprobación de los errores obtenidos en el cálculo de las coordenadas de las balizas.

3. Pliego de condiciones generales

En esta sección se describen los contenidos del proyecto, la ejecución del mismo, las relaciones entre el contratante y contratista y los aspectos administrativos a tener en cuenta para el desarrollo del mismo.

3.1. Documentación del proyecto

La documentación generada en el proyecto es la siguiente:

- Pliego de condiciones.

- Presupuesto.
- Memoria del proyecto.
- Manual del usuario.

En la memoria del proyecto se recogerá la descripción del sistema de simulación desarrollado, así como los resultados obtenidos. Para la modificación de este, el contratante deberá referirse al manual del usuario, proporcionado como anejo del proyecto.

3.2. Condiciones Generales Facultativas

En este apartado se detallan las relaciones entre el contratante y el contratista.

Funciones del contratante

El contratante es responsable de aportar la siguiente información para el desarrollo de este proyecto:

- Modelo matemático de la aeronave simulada, correspondiente a la información geométrica y aerodinámica requeridas para la descripción completa de la misma.
- Sistema de control. El contratante debe proveer la estructura del sistema de control que desee utilizar, o en su defecto, la estructura y parámetros estimados. Tras la finalización del proyecto, el contratante podrá modificar estos según sus necesidades.

Funciones del contratista

El contratista se compromete a la realización de las partes de este proyecto, proporcionando al final del mismo los documentos indicados en el apartado 3.1. Concretamente, deberá proporcionar un manual de usuario en el que se especifiquen la soluciones a posibles problemas que puedan aparecer durante el uso del sistema de simulación, de la misma manera en que proporcione información clara y concisa que permita la modificación y adecuación del mismo según las necesidades del contratante varíen.

Por otro lado, no garantiza el correcto funcionamiento del sistema de control o la exactitud del modelo dinámico proporcionado, siendo estas las competencias del contratante.

3.3. Condiciones de la Ejecución del Proyecto

El tiempo de ejecución acordado entre el contratante y el contratista figura en el pliego de prescripciones técnicas, donde se detallan las etapas de desarrollo y se deberá dar cuenta por escrito si así lo solicita el contratante.

Recepción del proyecto

La documentación generada en este proyecto se entregará íntegramente al contratante. En ella se dispondrá de los documentos especificados en el apartado 3.1, así como el código desarrollado en MATLAB[®] y Simulink[®].

El código proporcionado será plenamente funcional sin necesidad de modificación alguna por parte del contratante, en una primera instancia. Si el contratante no está conforme con el sistema de simulación desarrollado, podrá solicitar su modificación siempre y cuando corresponda con el alcance de este proyecto. Si el motivo de su disconformidad son las características del sistema de control o la precisión del modelo dinámico, será su competencia la modificación de estos, no correspondiendo con la labor del contratista.

Asimismo, en caso de que el sistema de simulación se ejecute en otro ordenador diferente al utilizado para su desarrollo, no corresponde al contratista proporcionar los programas necesarios. Estos programas se encuentran detallados en el pliego de prescripciones técnicas, y corresponde al usuario final su instalación en cualquier caso.

Ampliaciones y prórrogas

Si por causas de fuerza mayor no es posible iniciar o finalizar una parte de este proyecto en el plazo estipulado, se otorgará una prórroga para el cumplimiento del contrato. En cualquier caso, esta prórroga deberá ser consensuada entre el contratante y el contratista.

3.4. Condiciones Generales Económicas

En este apartado se detallan las condiciones económicas del proyecto, estando su desarrollo completo en el documento del presupuesto.

Precios

El cálculo de precios resulta de la suma de costes directos, gastos generales y el beneficio industrial:

- Costes directos: incluyen el precio de la mano de obra, el precio del material utilizado y el precio de las licencias de software de los programas empleados.
- Gastos generales: incluyen gastos que no se tienen en cuenta en los costes directos, como el gasto energético. Estos gastos se sitúan en un 13% de los costes directos.
- Beneficio industrial: se considera un valor del 6% de los costes directos.

4. Pliego de prescripciones técnicas

4.1. Objeto

El objeto del pliego de prescripciones técnicas es describir las características de los materiales utilizados en el desarrollo del proyecto, donde se incluyen todas las licencias de software necesarias, así como las condiciones de ejecución de los mismos.

4.2. Condiciones de los materiales

Los materiales utilizados en este trabajo corresponden a los programas informáticos necesarios para el desarrollo del sistema de simulación. A continuación se detallan las condiciones de uso de cada uno:

- MATLAB[®] y Simulink[®]: se emplea la versión R2020b de estos programas como base del sistema de simulación, con la versión 10.2 de Simulink[®]. La licencia estándar de este producto es de uso individual, pudiéndose instalar en un total de cuatro ordenadores. Además, es necesaria la instalación de los siguientes complementos, que incluyen las funciones y bloques utilizados en el sistema de simulación:
 - *Aerospace Toolbox* - versión 3.4
 - *Aerospace Blockset* - versión 4.4
 - *Symbolic Math Toolbox* - versión 8.6
 - *UAV Toolbox* - versión 1.0
 - *Navigation Toolbox* - versión 1.2
 - *Computer Vision Toolbox* - versión 9.3
 - *Image Processing Toolbox* - versión 11.2
 - *Control System Toolbox* - versión 10.9
- Unreal[®] Engine: permite la distribución del código creado al cliente, en la forma de aplicaciones en formato ejecutable. El cliente tiene derecho a su uso, reproducción, mostrar y ejecutar públicamente el producto. Sin embargo, el cliente no tiene derecho a la distribución del mismo. La información completa sobre las condiciones de uso de este programa se encuentran en el apartado EULA (*End User License Agreement*) de su página web. Se emplea la versión 4.23 de este motor de juegos.
- QGIS: es un programa de Sistemas de Información Geográfica (SIG) de Código Abierto licenciado bajo *GNU - General Public License* (GNU GPL), que garantiza a usuarios finales la libertad de usar, estudiar, compartir y modificar el software. La versión empleada es la 3.18.3.
- GIMP: es un programa de edición de imágenes digitales sujeto a la licencia GNU GPL y *GNU Lesser General Public License* (GNU LGPL). La licencia GNU LGPL incluye un conjunto de permisos adicionales a la licencia GNU GPL. La versión empleada es la 2.10.
- QGroundControl: es una estación de control terrestre, que permite la planificación de la misión autónoma en el sistema de simulación desarrollado. Se encuentra protegido bajo la licencia GPLv3 y Apache 2.0, la cual requiere la conservación del aviso de derecho de autor. La versión empleada es la v4.1.1.

- Overleaf: editor de texto online, utilizado para crear los documentos de este proyecto. Las condiciones de uso de este programa se encuentra en el apartado legal de su página web ¹.

Respecto a las referencias bibliográficas utilizadas en este proyecto, aquellas que no sean de libre acceso serán consultadas mediante la RedIRIS, que es la red de recursos informáticos española, disponible para miembros de la Universitat Politècnica de València.

4.3. Condiciones de ejecución

En este apartado se detalla el uso de los materiales explicados anteriormente, para cumplir las partes establecidas en este proyecto:

- Modelo dinámico y sistema de control de la aeronave: el modelo dinámico y el sistema de control se implementan en Simulink[®], haciendo uso de las herramientas *Aerospace Toolbox*, *Aerospace Blockset*, *Symbolic Math Toolbox* y *Control System Toolbox*. Esta implementación corresponde con las ecuaciones de Bryan, que aproximan la respuesta de una aeronave en vuelo real.
- Creación del entorno de visualización realista: el entorno de visualización realista se crea mediante el editor de Unreal Engine. Los archivos necesarios para su creación se obtienen mediante el programa QGIS, donde se realiza el tratamiento de información geográfica para representar el mundo real. La conexión con Simulink[®] se realiza mediante los bloques proporcionados en la herramienta *UAV Toolbox*.
- Creación de balizas terrestres en el entorno de visualización: se realiza únicamente dentro del editor de Unreal Engine. Las balizas son colocadas sobre el terreno, apuntando las coordenadas en las que se encuentran con el fin de obtener los datos necesarios para la validación del sistema.
- Conexión con la estación de control terrestre: la conexión se establece mediante Simulink[®] y *QGroundControl*. Es en esta estación de control terrestre en la cual se realiza la planificación de las misiones autónomas.
- Implementación de algoritmos de visión artificial: los algoritmos de visión artificial se implementan mediante las funciones proporcionadas por la herramienta *Computer Vision Toolbox*. Estas herramientas se utilizan tanto para el entrenamiento del modelo, como para la detección de objetos en imágenes.
- Cálculo de las coordenadas de las balizas: se implementa sobre funciones en Simulink[®], mediante algoritmos de elaboración propia. La detección de las balizas se realiza utilizando herramientas proporcionadas en *Image Processing Toolbox*.
- Validación del sistema: se realiza una vez finalizada la simulación de la misión autónoma, mediante la obtención de los errores en el cálculo de la posición de las balizas. Estos errores se utilizan para cuantificar la precisión del sistema, con lo que se validan los algoritmos de visión artificial y el cálculo de las coordenadas. El seguimiento de la trayectoria definida en la misión se verifica mediante la visualización del estado de la aeronave y la información mostrada en *QGroundControl*.

¹Disponible en: <https://www.overleaf.com/legal>

4.4. Pruebas y ajustes finales o de servicio

Tras la finalización de las partes del proyecto, el contratante puede verificar el sistema de simulación desarrollado mediante pruebas de diversas misiones autónomas. Si los resultados que obtiene tras estas pruebas no son satisfactorios, puede solicitar la modificación del sistema de simulación. Esta fase correspondería con los ajustes finales del proyecto.

Finalmente, el contratante será proporcionado con toda la documentación generada en el proyecto, junto con el código desarrollado en cada parte, según las condiciones de ejecución.



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



TRABAJO FIN DE GRADO

Diseño de un sistema de simulación de misiones
autónomas para sistemas aéreos no tripulados.
Aplicación al proyecto HORUS UPV

Presupuesto

AUTOR

GARCÍA CASES, Arnau argarca@etsid.upv.es

TUTOR

GARCÍA-NIETO RODRÍGUEZ, Sergio sgnieto@isa.upv.es

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO

GRADO EN INGENIERÍA AEROESPACIAL
CURSO 2020-2021

Índice

1	Introducción	1
2	Presupuestos parciales	1
2.1	Mano de obra	1
2.2	Material	2
2.3	Licencias de software	3
3	Presupuesto global	4

1. Introducción

En este presupuesto se desglosan los costes para llevar a cabo el Trabajo Fin de Grado “Diseño de un sistema de simulación de misiones autónomas para sistemas aéreos no tripulados. Aplicación al proyecto HORUS UPV”.

El cálculo de costes parciales diferencia entre los precios de la mano de obra, los precios del material utilizado y los precios de las licencias de software. El presupuesto global es la suma de los parciales, al que se debe añadir un 13% debido a gastos generales, un 6% de beneficio industrial y el 21% del IVA.

2. Presupuestos parciales

Para el cálculo de los presupuestos parciales se tiene en cuenta el tiempo dedicado a cada etapa del proyecto. Estas etapas se detallan en el pliego de condiciones y se resumen a continuación, junto con el software empleado en cada una de ellas:

1. **P1:** modelo dinámico y sistema de control de la aeronave. El software empleado es: MATLAB, Simulink, Aerospace Toolbox, Aerospace Blockset, Symbolic Math Toolbox, UAV Toolbox, Control System Toolbox y Navigation Toolbox.
2. **P2:** creación del entorno de visualización realista. El software empleado es: Unreal Engine, Epic Games Launcher, QGIS y GIMP.
3. **P3:** creación de las balizas terrestres en el entorno de visualización. El software empleado es: Unreal Engine y Epic Games Launcher.
4. **P4:** conexión con la estación de control terrestre. El software empleado es: MATLAB, Simulink y QGroundControl.
5. **P5:** implementación de algoritmos de visión artificial. El software empleado es: MATLAB, Simulink, Image Processing Toolbox y Computer Vision Toolbox.
6. **P6:** cálculo de las coordenadas de las balizas. El software empleado es: MATLAB, Simulink, Image Processing Toolbox y Computer Vision Toolbox.
7. **P7:** validación del sistema. El software empleado es: MATLAB, Simulink, Aerospace Toolbox, Aerospace Blockset, Symbolic Math Toolbox, UAV Toolbox, Control System Toolbox, Navigation Toolbox, Computer Vision Toolbox, Image Processing Toolbox y QGroundControl.

Por otro lado, en el cálculo del coste del material, la mano de obra y las licencias de software se tiene en cuenta el número total de horas laborables anuales. En este caso se consideran 8 horas diarias, durante 21 días al mes y 12 meses por año, por lo que **el total de horas laborables anuales es de 2016 h.**

2.1. Mano de obra

Los costes de mano de obra corresponden con la labor del ingeniero junior, cuyo coste por hora se calcula de acuerdo a la disposición 14977 del BOE núm. 251 de 2019, por la que se registra

y publica el XIX Convenio colectivo del sector de empresas de ingeniería y oficinas de estudios técnicos. En el Anexo I de la misma se establece un salario total anual de 20 424.25 €, correspondiente al año 2020 y el nivel 2 (DIPLOMADOS Y TITULADOS 1^{er} CICLO UNIVERSITARIO. JEFE SUPERIOR). Dividiendo el salario total anual entre el total de horas laborables, se obtiene un coste por hora de 10.14 €, redondeando los céntimos al alza.

El presupuesto total de la mano de obra se recoge en la Tabla 1, donde se tiene en cuenta el tiempo empleado en cada fase del proyecto, junto con la redacción del mismo, así como el coste por hora establecido anteriormente.

Etapa del proyecto	Duración (horas)	Total [€]
P1	55	557.70
P2	60	608.40
P3	12	121.68
P4	18	182.52
P5	30	304.20
P6	16	162.24
P7	24	243.36
Redacción	105	1064.70
SUBTOTAL		3244.80

Tabla 1. Costes de la mano de obra

2.2. Material

El coste del material corresponde con el ordenador que se ha utilizado para el desarrollo del sistema de simulación. En concreto, se trata del *HP Envy 14-eb0002ns*, con un procesador *i7-1165G7 Quad Core*, con 16 GB de memoria RAM y 1 TB de almacenamiento SSD. En el cálculo del coste del material se considera un periodo de amortización de 4 años, por lo que el número de horas totales de utilización del material son 8064 h. Teniendo en cuenta la utilización del mismo en este proyecto, el coste es el dado en la Tabla 2.

Material	HP ENVY 14-eb0002ns
Coste de adquisición [€]	1287.08
Horas de utilización en el proyecto	320
Horas de utilización totales	8064
Coste total [€]	51.07

Tabla 2. Coste asociado al material

Las referencias bibliográficas utilizadas en este proyecto son de libre acceso o pueden ser consultadas mediante la RedIRIS, por lo que no hay coste asociado a la utilización de las mismas.

2.3. Licencias de software

Para este proyecto se considera únicamente un empleado trabajando en el diseño del sistema de simulación, por lo que solo se considera el precio de una única licencia de software para cada programa. El coste de cada licencia se amortiza según las horas de utilización de las mismas, teniendo en cuenta el precio anual de la licencia y el total de horas laborables anuales. Los resultados se muestran en la Tabla 3, donde las licencias de los productos de MathWorks[®] corresponden con el precio estándar anual dado en su página web¹.

Licencia	Coste anual [€]	Utilización [h]	Coste proyecto [€]
MATLAB [®]	800	143	56.75
Simulink	1200	143	85.12
Aerospace Toolbox	500	79	19.59
Aerospace Blockset	500	79	19.59
Symbolic Math Toolbox	400	79	15.67
UAV Toolbox	800	79	31.35
Navigation Toolbox	680	79	26.65
Computer Vision Toolbox	500	70	17.36
Image Processing Toolbox	400	70	13.89
Control System Toolbox	460	79	18.03
Unreal Engine	0	72	0
Epic Games Launcher	0	72	0
QGIS	0	15	0
GIMP	0	10	0
QGroundControl	0	42	0
Overleaf	0	105	0
TOTAL			304.00

Tabla 3. Coste asociado a las licencias de software

¹Disponible en: <https://es.mathworks.com/pricing-licensing.html>

3. Presupuesto global

Los costes totales vienen dados por la suma de los costes de material, los costes de las licencias de software y los costes de mano de obra. En la Tabla 4 se resume el total de los presupuestos parciales, sin aplicar el IVA, los gastos generales o el beneficio industrial.

Tipo de coste	TOTAL [€]
Mano de obra	3244.80
Material	51.07
Licencias	304.00
TOTAL [€]	3599.87

Tabla 4. Coste total bruto

El presupuesto base de licitación se obtiene tras aplicar los beneficios industriales, los gastos generales, y el IVA. El cálculo se resume en la Tabla 5.

	TOTAL [€]
COSTE TOTAL BRUTO	3599.87
13 % gastos generales	467.98
6 % beneficio industrial	215.99
PRESUPUESTO DE INVERSIÓN	4283.85
21 % IVA	899.61
PRESUPUESTO BASE DE LICITACIÓN	5183.46

Tabla 5. Presupuesto total de licitación

Por tanto, el presupuesto final de este Trabajo Fin de Grado, asciende a la cantidad de EUROS:

CINCO MIL CIENTO OCHENTA Y TRES con CUARENTA Y SEIS



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



TRABAJO FIN DE GRADO

Diseño de un sistema de simulación de misiones
autónomas para sistemas aéreos no tripulados.
Aplicación al proyecto HORUS UPV

Manual de usuario

AUTOR

GARCÍA CASES, Arnau argarca@etsid.upv.es

TUTOR

GARCÍA-NIETO RODRÍGUEZ, Sergio sgnieto@isa.upv.es

UNIVERSITAT POLITÈCNICA DE VALÈNCIA
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DEL DISEÑO

GRADO EN INGENIERÍA AEROESPACIAL
CURSO 2020-2021

Índice general

1	Introducción	1
1.1	Estructura del sistema de simulación	2
2	Sistema aéreo no tripulado (UAS)	5
2.1	Modelo dinámico	5
2.2	Sistema de control	6
2.2.1	Lazo externo	7
3	Estación de control terrestre	9
4	Escenario de visualización realista	13
4.1	Obtener de los datos necesarios para crear el escenario	14
4.1.1	Modelo Digital del Terreno	14
4.1.2	Imagen satelital	15
4.2	Crear el mapa en Unreal Engine	16
4.2.1	Importar el mapa de alturas	17
4.2.2	Importar la imagen satelital como textura	18
4.3	Crear las balizas terrestres	19
4.4	Empaquetar el proyecto	20
5	Visión artificial	23
5.1	Detección de objetos	23
5.1.1	Entrenamiento	24
5.1.2	Evaluación	25
5.1.3	Implementación en <i>Simulink</i>	25
5.2	Cálculo de las coordenadas	26
5.3	Análisis de resultados	27

1 | Introducción

En este documento se describe el sistema de simulación desarrollado, de modo que el usuario pueda modificarlo según sus necesidades. Para ello, se estructura en un total de cuatro capítulos con el siguiente contenido:

1. Sistema aéreo no tripulado (UAS): describe los parámetros modificables por el usuario, para ajustar tanto el modelo dinámico como el sistema de control, que incluye el lazo de control externo y el lazo de control interno.
2. Estación de control terrestre: describe el proceso de crear el plan de vuelo en *QGroundControl* y su utilización durante la simulación.
3. Escenario de visualización realista: explica en detalle el proceso de creación de un mapa nuevo, a partir del proyecto en Unreal Engine proporcionado. Esto incluye el procesamiento de los datos necesarios para crearlo, el proceso de creación en Unreal Editor, junto con las balizas necesarias para el algoritmo de visión artificial, y finalmente el empaquetado del proyecto para ser utilizado en el sistema de simulación.
4. Visión artificial: incluye la obtención de los vídeos para entrenar el detector de balizas, el cálculo de las coordenadas y el análisis de los resultados tras la simulación.

El código del proyecto al que se hace referencia en este manual se encuentra en una misma carpeta comprimida (*Sistema_de_simulacion.zip*). Dentro de esta carpeta se encuentran diversos archivos y otras carpetas, a las que se hará referencia a lo largo del documento. En concreto, los principales archivos son:

- *MasterScript.m*: en este se definen todas las variables utilizadas por el archivo de Simulink. Al iniciar MATLAB, este **script** debe ser ejecutado antes de ejecutar el archivo de Simulink. Entre simulaciones consecutivas no es necesario ejecutarlo si no se ha modificado este u otros archivos.
- *HORUS_control.slx*: es el archivo de Simulink que contiene el sistema de simulación. Al abrirlo pueden observarse 3 subsistemas (UAS, Animación y Estación de control terrestre), a los que se puede acceder haciendo doble clic sobre ellos.

En el sistema de simulación desarrollado se encuentran varios subsistemas variantes, mediante los cuales se puede elegir las partes del sistema que deben ejecutarse, a través de unas variables definidas en el **workspace** de MATLAB. Estas variables están definidas en la última sección del **script** *MasterScript.m*:

- *isPhotoRealisticSim*: permite activar la visualización en el entorno realista (descrito en el capítulo 4), si el valor de la misma es igual a 1. En caso de ser igual a 0, la posición de la aeronave se visualizará en una gráfica de MATLAB.
- *showVideoViewer*: si es igual a 1 se visualizará la imagen captada por la cámara en el entorno realista (para ello la visualización en el entorno realista tiene que estar activada).
- *useCV*: se utiliza para activar o desactivar los algoritmos de visión artificial. Es de especial utilidad ya que el uso de visión artificial ralentiza la simulación, por lo que se recomienda desactivarla cuando no sea necesario su uso. Cabe destacar que el uso de visión artificial solo es posible si las variables *isPhotoRealisticSim* y *showVideoViewer* son iguales a 1.
- *conectarQGC*: determina si la misión considerada es la definida en el **script** *MasterScript.m* o la definida en *QGroundControl*, cuyo uso se detalla en el capítulo 3.

En los capítulos de este documento se detalla el funcionamiento de los sistemas únicamente cuando estos están activados. Por último, no se asegura el funcionamiento del sistema de simulación en el caso de que las versiones de los programas instalados difieran de aquellas especificadas en el pliego de condiciones de este proyecto.

1.1. Estructura del sistema de simulación

La estructura general del sistema de simulación es la dada en la Figura 1.1. El subsistema **UAS** se explica en detalle en el capítulo 2, mientras que el subsistema **Estación de control terrestre** se explica en el capítulo 3.

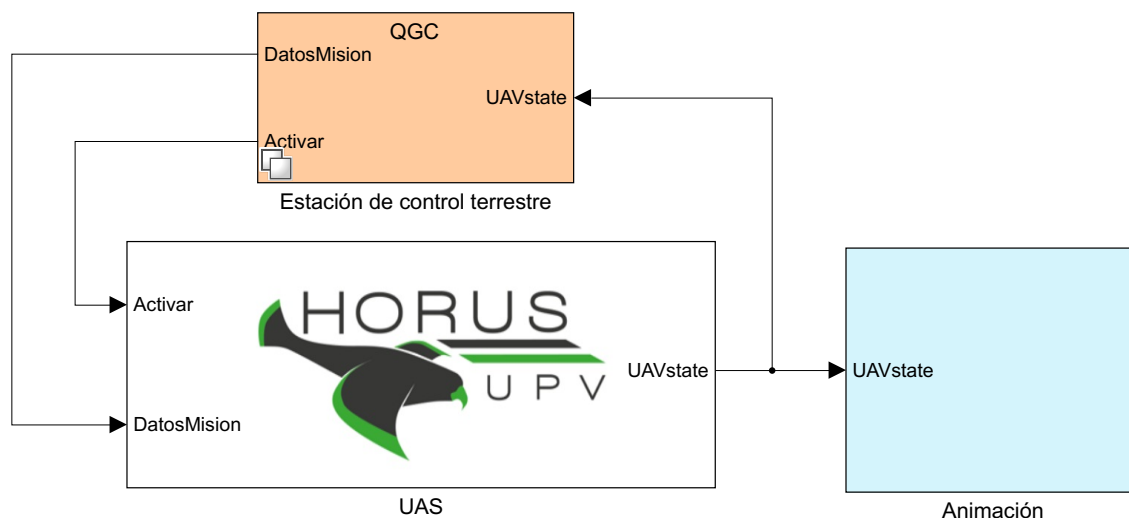


Figura 1.1. Esquema general de la implementación en Simulink

Por otro lado, dentro del subsistema de animación se encuentra la implementación tanto de la visualización de la misión en el entorno realista como de los algoritmos de visión artificial. El esquema de este subsistema puede observarse en la Figura 1.2, donde el subsistema **Tipo de simulación** se explica con detenimiento en el capítulo 4 y el subsistema **Visión artificial** en el capítulo 5.

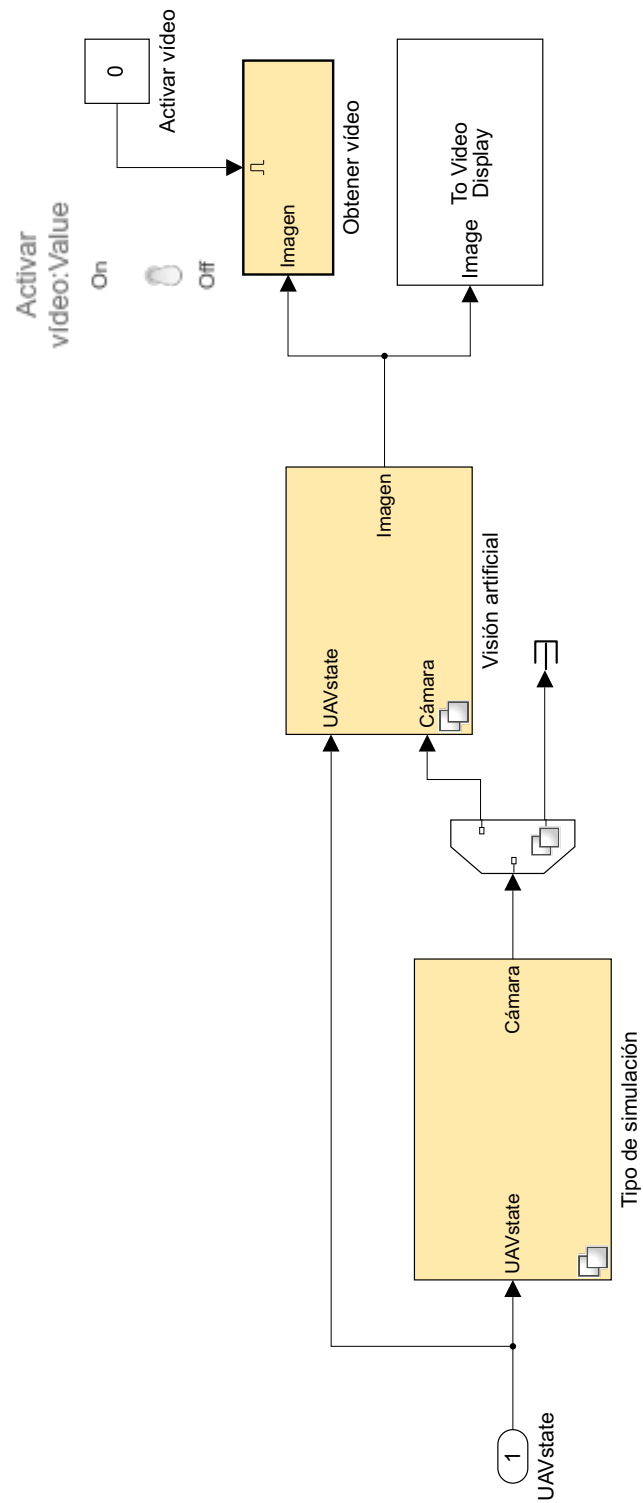


Figura 1.2. Esquema general del subsistema *Animación*

2 | Sistema aéreo no tripulado (UAS)

La estructura general del sistema aéreo no tripulado se muestra en la Figura 2.1. Los subsistemas de **Entorno** y **Modelo dinámico** se explican en la sección 2.1, mientras que los subsistemas **Guiado** y **Sistema de control** se explican en la sección 2.2.

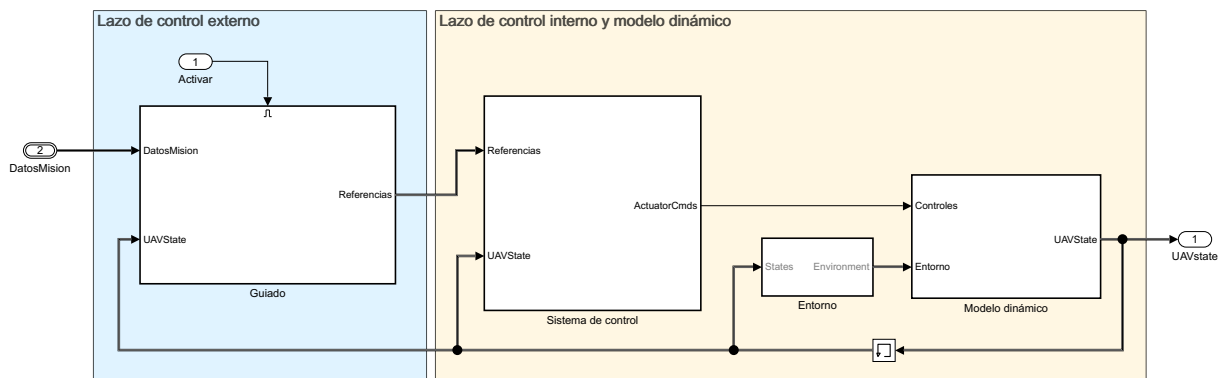


Figura 2.1. Esquema general del bloque *UAS*

2.1. Modelo dinámico

El modelo dinámico detallado en la memoria del proyecto se implementa en el subsistema **HORUS_control > UAS > Modelo dinámico**, con la estructura dada en la Figura 2.2. El bloque principal, *6 DOF Equations of Motion*, utiliza los valores iniciales definidos en la sección *Condiciones Iniciales - parámetros de simulación* del archivo *MasterScript.m*, y las características de la aeronave, disponibles en **ArchivosDeDatos/Horus_data.mat**.

Si se desea modificar los parámetros del modelo dinámico de la aeronave, es necesario modificar y ejecutar el archivo *Data_Generation.m*, que se encuentra dentro del directorio principal del código de este proyecto. En este archivo se encuentran todos los coeficientes que intervienen en el cálculo de fuerzas y momentos que actúan sobre la aeronave, así como las ecuaciones de Bryan que describen su comportamiento.

Dentro del modelo dinámico se encuentra también el subsistema para crear la estructura del bus de datos. Este bus se utiliza para organizar todas las variables que describen el estado de la aeronave, de forma que un error en el sistema pueda ser detectado con mayor facilidad. Los buses utilizados deben ser creados en MATLAB, y esto es lo que hace el archivo *BusDef.m* en el directorio principal del proyecto. No se debe modificar este subsistema ni el archivo donde se definen los buses utilizados, ya que una incorrecta modificación provocaría un fallo en el sistema

de simulación.

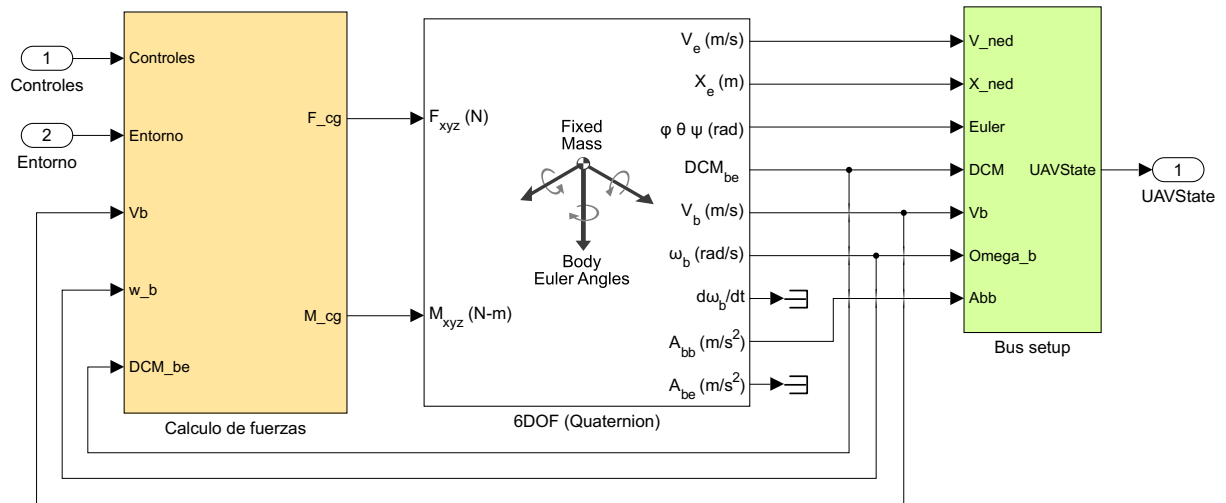


Figura 2.2. Implementación del modelo dinámico

Por otro lado, como se explica en la memoria del proyecto, existe la posibilidad de incorporar viento a la simulación. Para ello, en el subsistema **HORUS_control > UAS > Entorno** hay que modificar la intensidad del ruido blanco de los bloques de la Figura 2.3. El viento se especifica en el sistema de referencia NED, por lo que hay que generar el viento en sus tres componentes individuales (w_{vx} , w_{vy} , w_{vz}). Por ejemplo, si se establece una potencia de ruido de 0.1 para un eje determinado, la magnitud del viento en el mismo alcanza picos de 3 m s^{-1} .

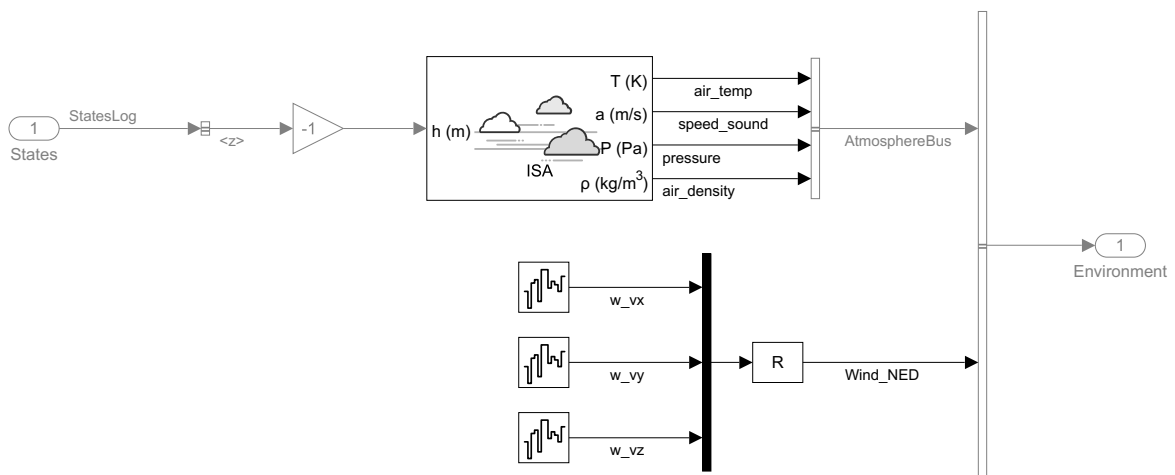


Figura 2.3. Implementación del subsistema *Entorno*

2.2. Sistema de control

El sistema de control, dividido en el lazo interno y el lazo externo, se encuentra implementado en el subsistema UAS del archivo de Simulink. Los controladores PID del mismo se implementan mediante el bloque *PID controller* de Simulink, como se muestra en la Figura 2.4, dentro del cual se tienen que definir las ganancias del controlador (pestaña *Main*). Asimismo, la saturación

de la salida se define en la pestaña *Output Saturation*, lo cual debe hacerse teniendo en cuenta las limitaciones estructurales y aerodinámicas de la aeronave.

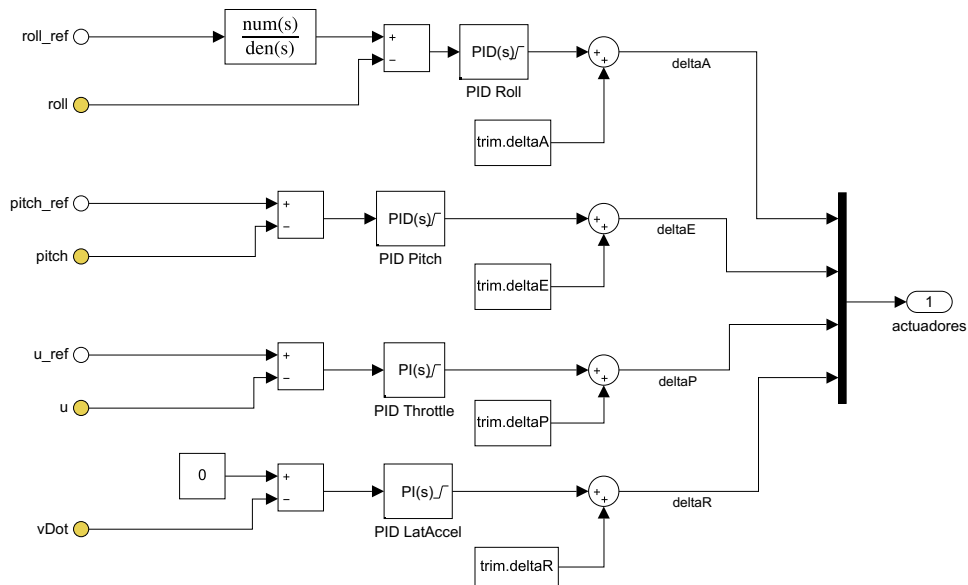


Figura 2.4. Implementación de los controladores del lazo de control interno, dentro del subsistema *Sistema de control*

Por defecto, los controladores de las superficies de control (alergones, elevador y timón de cola) están limitados a $\pm 30^\circ$, mientras que los controladores del ángulo de alabeo y el ángulo de cabeceo están limitados a $\pm 60^\circ$ y $\pm 15^\circ$, respectivamente. Cabe destacar que las salidas de los controladores corresponden a incrementos respecto al punto de funcionamiento, por lo que al valor máximo de las acciones de control se tiene que restar el valor del punto de funcionamiento. De esta forma, tras sumar a la acción de control el valor máximo de las superficies o ángulos, estos no exceden los límites estructurales o aerodinámicos.

Los valores de saturación de los bloques PID deben ser modificados directamente desde Simulink, abriendo el bloque (doble clic sobre el mismo) y especificando el valor en la pestaña *Output saturation*. Por el contrario, los valores de las ganancias de los controladores deben ser modificados en el archivo *Controladores.m*, antes de ejecutar el archivo *MasterScript.m*.

2.2.1. Lazo externo

El lazo de control externo es activado en el momento en que se sube la misión mediante la estación de control terrestre, o al inicio de la simulación si se desactiva el subsistema que incluye la conexión con esta. Dentro del mismo, en el subsistema **HORUS_control** > **UAS** > **Guiado** > **Guiado**, mostrado en la Figura 2.5, se encuentran los parámetros modificables del algoritmo de guiado.

En la constante *Lookahead Distance* debe elegirse la distancia adecuada, haciendo doble clic sobre la misma y modificando el parámetro, según las características de la aeronave y la respuesta del lazo de control interno. La explicación de este parámetro se encuentra en la memoria del proyecto.

Por otro lado, en los parámetros del bloque *Waypoint Follower* (doble clic sobre el mismo) se especifica el radio de transición y la distancia sobre la trayectoria mínima para calcular el

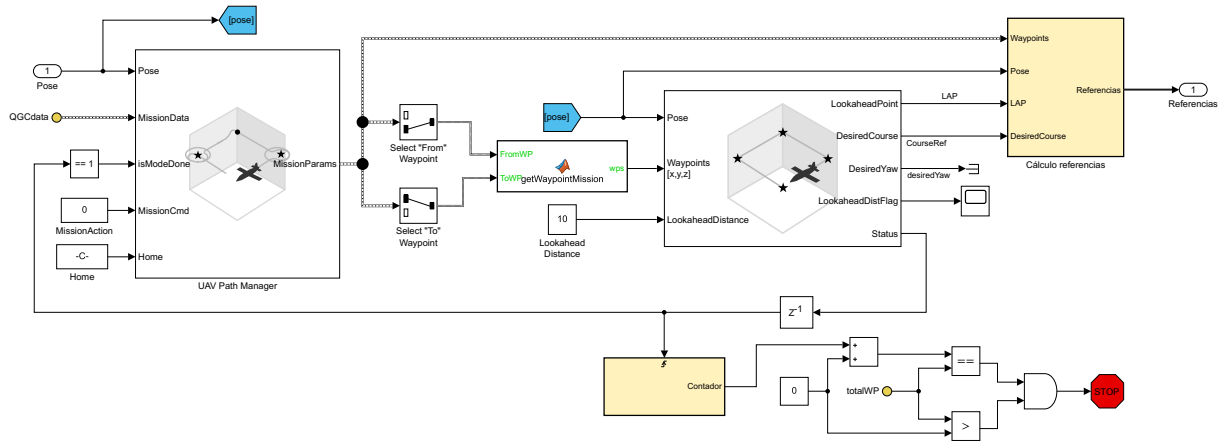


Figura 2.5. Implementación del bloque de guiado

LAP (*Lookahead Point*). El usuario puede experimentar con variaciones en estos parámetros y determinar cuál es el valor adecuado para obtener una respuesta deseada. Por último, en el subsistema **Cálculo referencias**, cuya implementación se da en la Figura 2.6, se encuentran los controladores PID en los cuales se puede especificar la saturación de la salida y cuyas ganancias se especifican en el archivo *Controladores.m*.

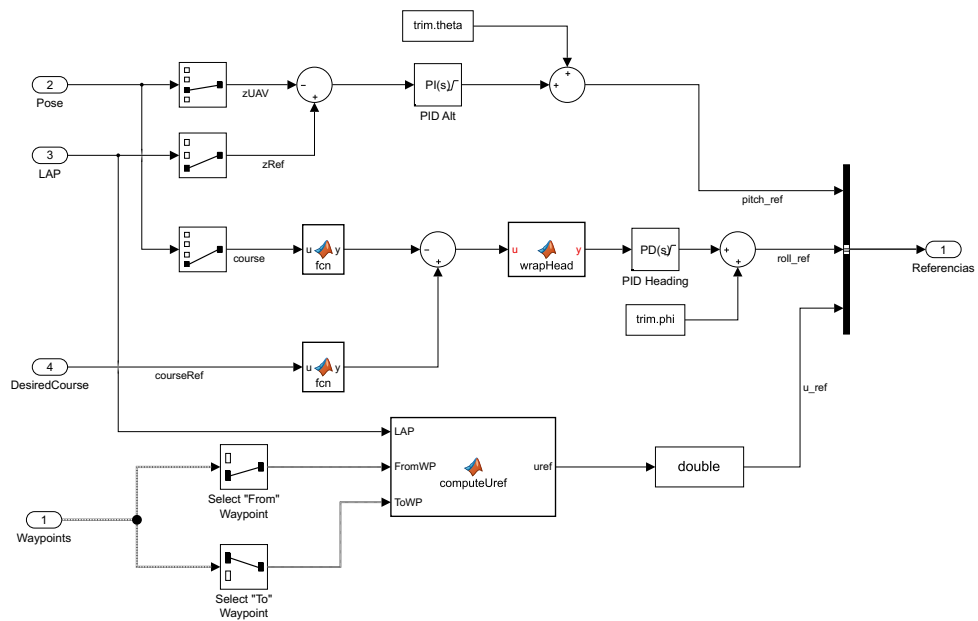


Figura 2.6. Cálculo de las referencias utilizadas por el lazo de control interno

3 | Estación de control terrestre

La conexión de Simulink y la estación de control terrestre (*QGroundControl*) está implementada en el subsistema **HOURS_control > Estación de control terrestre > QGC**, según se muestra en la Figura 3.1. Los datos enviados a QGC corresponden con el estado de la aeronave (posición y orientación), mientras que los datos recibidos corresponden al número total de waypoints, la información de cada waypoint y la variable para indicar que la misión se ha enviado correctamente.

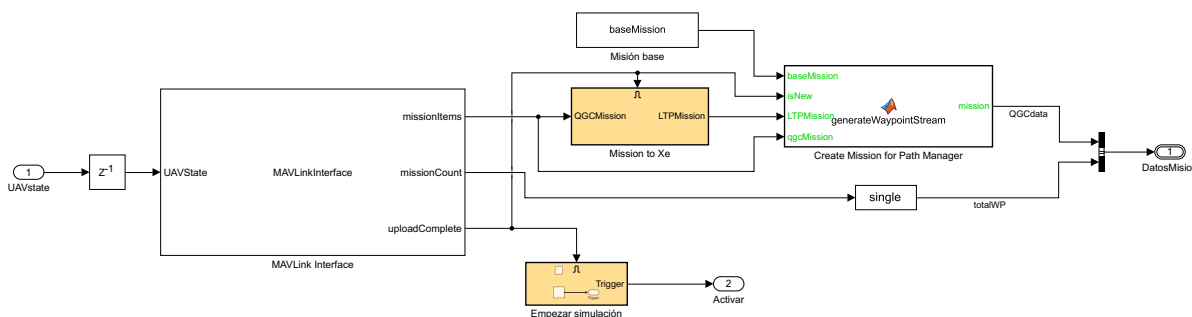





Figura 3.1. Conexión con QGC en Simulink

La misión base utilizada en este subsistema se define en el archivo *MasterScript.m*, mediante la variable *WP_List*. Esta variable contiene la latitud, longitud, altura y velocidad de cada waypoint. Si no se activa el subsistema de conexión con QGC, esta será la misión utilizada por el sistema de simulación. Por el contrario, si se activa el subsistema de comunicación con QGC, esta misión se utilizará de base para añadir la creada con la estación de control terrestre.

Una vez abierto QGC, el plan de vuelo puede crearse de la siguiente forma:

- Seleccionar la opción de plan de vuelo, mediante la herramienta 
- Crear el punto de despegue, mediante la herramienta 
- Definir los waypoints de la misión mediante la herramienta . Las características de los waypoints se establecen en la lista de comandos de la misión, como se muestra en la Figura 3.2. Si bien el primer parámetro indica que debe especificarse la altura, esta corresponde con la altura respecto al origen de coordenadas. El origen corresponde con el punto que indica el inicio de la misión (*Mission Start*), en cuyos atributos se indica por defecto que la altura es respecto al punto de lanzamiento (*Relative To Launch*). Es decir, si se indica que

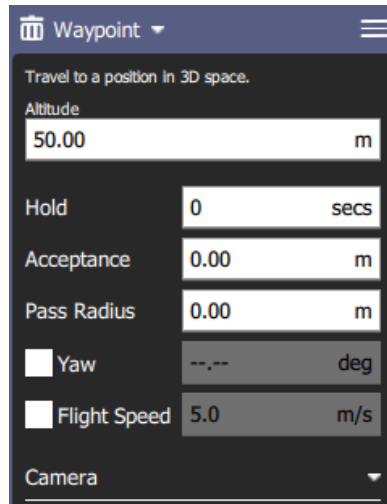


Figura 3.2. Parámetros de los waypoints en QGC

la “altitud”¹ es de 50 m quiere decir que la aeronave volará a una altura de 50 m respecto al suelo, en el origen de coordenadas.

Por otro lado, la velocidad de cada waypoint se establece haciendo clic sobre las tres líneas paralelas en la esquina superior derecha de la Figura 3.2 y seleccionando la opción *Show all values*. Con esto se tienen todas las características del waypoint, dentro de las cuales la velocidad se especifica en el parámetro 4 (*param4*), como se muestra en la Figura 3.3.

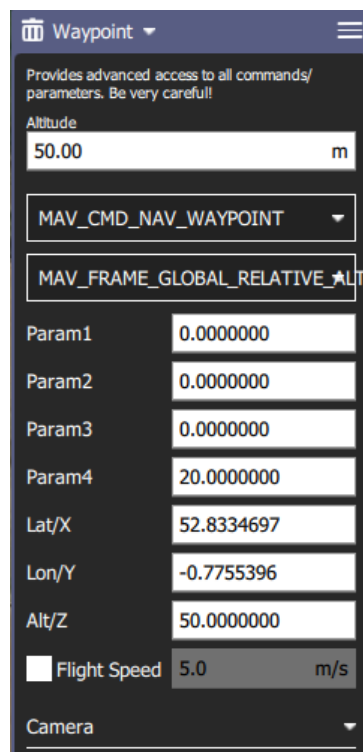


Figura 3.3. Establecer la velocidad de un waypoint en QGC

¹El uso de este término puede generar confusión, ya que por definición la altitud está referida a la distancia vertical de un punto respecto al nivel medio del mar. El término correcto aquí sería altura, ya que este indica la distancia vertical entre un punto y el suelo.

- Eliminar el punto de despegue (símbolo de la papelera en los parámetros del punto, dado en la esquina superior izquierda), ya que en el sistema de simulación la fase de despegue no está implementada, por lo cual la aeronave con una velocidad y altitud dadas. Este punto de despegue se crea para poder añadir los demás waypoints.

Posteriormente, habiendo activado el subsistema de conexión con la estación terrestre, al ejecutar el archivo de Simulink se establecerá automáticamente la conexión, haciendo uso de las clases definidas en el directorio principal del proyecto (*MAVLinkInterface.m*, *exampleHelperMAVMissionProtocol.m* y *exampleHelperMAVMissionTable.m*), que no deben ser modificadas por el usuario. En este caso la aeronave girará en torno al punto inicial con un ángulo de asiento lateral dado, como se muestra en la Figura 3.4, hasta que la misión sea subida.

El radio de giro inicial de la aeronave se determina según la estructura definida en la línea 61 de *MasterScript.m* (*initialReferences*). Si se aumenta el ángulo de alabeo (*roll_ref*), disminuye el radio de giro. No obstante, estos parámetros no son relevantes para la misión ya que se incluye de forma temporal hasta que la misión esté definida.

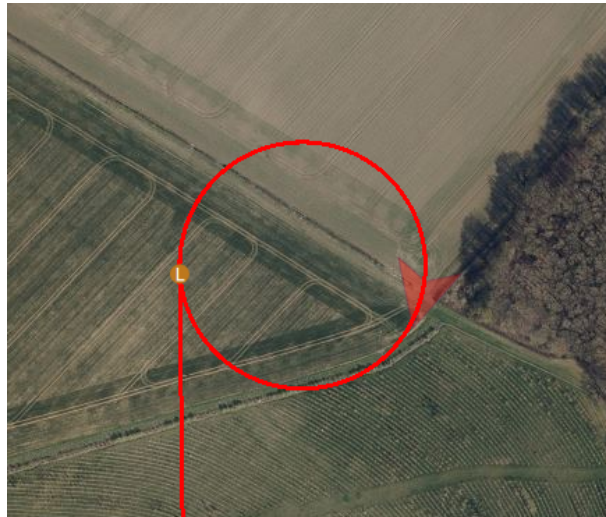
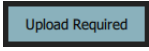


Figura 3.4. Trayectoria inicial de la aeronave al establecer la conexión con QGC

Finalmente, para subir la misión basta con hacer clic en el botón . Una vez se haya subido, el sistema de control será activado y la aeronave empezará a seguir la ruta definida hasta llegar al último waypoint.

Junto al código proporcionado en este proyecto se encuentra el directorio **QGC/**, en el cual existen tres misiones definidas en QGC. Estas misiones corresponden con las misiones para generar el vídeo de entrenamiento y evaluación del algoritmo de detección de objetos, junto con la misión final utilizada para analizar el resultado.

4 | Escenario de visualización realista

En el presente capítulo se explica detalladamente el proceso de creación de un mapa en Unreal Editor, así como su empaquetado para poder ser utilizado en Simulink. En caso de utilizar el mapa creado en este proyecto, las secciones 4.1 y 4.2 pueden ser omitidas. Estas se incluyen solo para aquellos usuarios que deseen crear un mapa distinto.

En cualquier caso, después de instalar Unreal Editor se deben seguir los siguientes pasos de forma que se pueda trabajar con el mapa proporcionado en este proyecto¹:

1. Instalar el paquete de ayuda para personalizar los escenarios: se instala desde la pestaña de **HOME** > **Add-Ons** > **Get Add-Ons** y buscando el paquete en cuestión (*UAV Toolbox Interface for Unreal Engine Projects*).

2. Añadir complementos a Unreal Engine: con el paquete de ayuda vienen instalados dos complementos que son necesarios para visualizar la aeronave en el escenario creado con Unreal Editor. Estos complementos se encuentran en la ruta:

```
C:\ProgramData\MATLAB\SupportPackages\R2020b\toolbox\uav\spkg\uavunrealengine\
mwas_plugins
```

Los dos directorios dentro de esta ruta, *MathWorksUAV* y *MathWorksSimulation*, deben ser copiados y pegados en el directorio de complementos de Unreal Engine. Por ejemplo, pueden ser copiados en el directorio de la siguiente ruta:

```
C:\ProgramFiles\EpicGames\UE_4.23\Engine\Plugins\MathWorks
```

El directorio de *MathWorks* se crea para agrupar ambos complementos, pero no es necesario.

Además de los complementos, también viene incluido un proyecto que sirve como base para la creación de nuevos mapas. Concretamente, este es el archivo utilizado para crear el entorno de visualización realista del sistema de simulación. No obstante, el usuario dispone de un archivo comprimido con el proyecto de Unreal Engine, bajo el nombre de *UnrealEngineMap*, proporcionado con el código de este trabajo.

Este proyecto debe ser abierto desde Simulink, haciendo doble clic sobre el bloque *Simulation 3D Scene Configuration*, que se encuentra en el subsistema **HORUS_control** > **Animación** > **Tipo de Simulación** > **Simulación realista**, dado en la Figura 4.1, y seleccionar la opción de Unreal Editor en el apartado *Scene source*. En la opción del proyecto se debe seleccionar el archivo con extensión *.uproject*², y posteriormente abrirlo mediante el botón *Open Unreal Editor*.

¹Los pasos aquí explicados son los más relevantes de la información proporcionada en la documentación de MATLAB, disponible en <https://es.mathworks.com/help/uav/ug/install-support-package-for-customizing-scenes.html>

²La ruta de acceso del archivo no debe contener caracteres especiales, tales como espacios o tildes. En caso de contener estos caracteres, se mostrará un error indicando que no ha sido posible abrir el archivo.

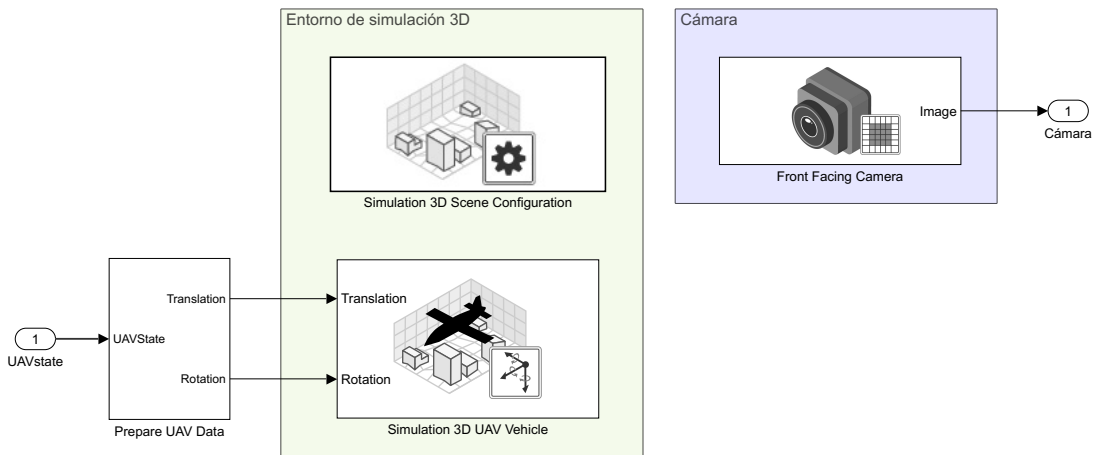


Figura 4.1. Implementación de la comunicación con el entorno realista

Una vez abierto el proyecto, seleccionar el menú **Editar > Plugins** y asegurarse de que los dos complementos estén activos dentro del proyecto. Se recomienda al usuario realizar el primer tutorial disponible en el programa para familiarizarse con la estructura del editor, haciendo clic sobre el birrete azul que se encuentra en la esquina superior derecha.

4.1. Obtener de los datos necesarios para crear el escenario

4.1.1. Modelo Digital del Terreno

El Modelo Digital del Terreno (MDT) puede ser descargado de diversas fuentes, según se indica en la memoria de este proyecto. Una vez se dispone del archivo, este puede ser abierto en un proyecto en blanco de QGIS simplemente arrastrando el archivo TIFF desde el explorador de archivos hasta el panel de capas (esquina inferior izquierda en la configuración de QGIS por defecto). Dependiendo de la fuente del archivo MDT, será necesario aplicar una transformación de SRC (Sistema de Referencia de Coordenadas) al SRC WGS84, que corresponde con el SRC por defecto en el proyecto en blanco. Esta transformación la detecta automáticamente QGIS, por lo que solo es necesario aceptar la operación. Asimismo, en el caso de que la transformación necesaria no estuviera instalada por defecto, QGIS indica los pasos que hay que seguir para instalar el paquete que contiene a la misma.

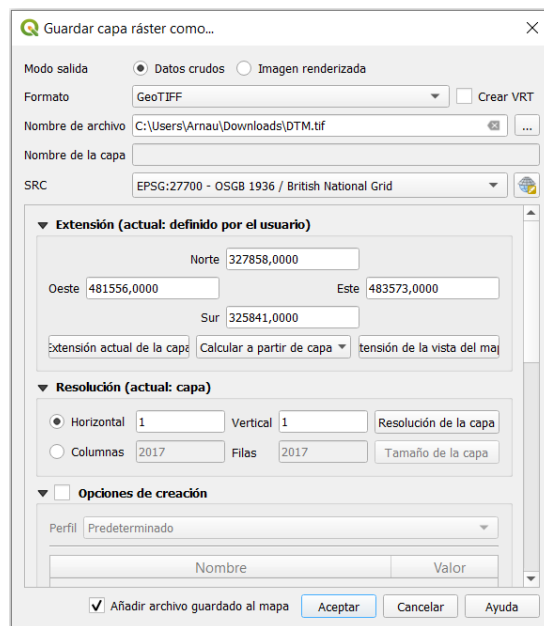


Figura 4.2. Recortar el MDT

El segundo paso, una vez importado el MDT, consiste en recortarlo para cumplir con la extensión compatible con Unreal Engine. Para rectorar la capa, se exporta el archivo según la Figura 4.2 (clic derecho sobre la capa > exportar

> guardar como). En el apartado extensión se define el tamaño de la capa que se quiere extraer, teniendo en cuenta que las coordenadas en el SRC especificado (*British National Grid*) están en metros. Con esto, se define una región de 2017 m × 2017 m. Por último, en el apartado de resolución se debe indicar la resolución de el archivo MDT descargado, siendo el caso del ejemplo una resolución de 1 m.

El último paso consiste en transformar el archivo MDT en un tipo de archivo que pueda ser importado en Unreal Engine. Esto es, hay que convertirlo en un mapa de altura. Para ello se utiliza el complemento *HeightmapExport*³, que puede instalarse desde la barra de menús, **Complementos > Administrar e instalar complementos**, buscando su nombre.

Tras instalar el complemento, se puede acceder al mismo desde la barra de menús, **Ráster > Heightmap Export > Heightmap Export**. En la ventana emergente que aparece se tienen que especificar las opciones de exportación, en concreto la altura y ancho de la imagen (número de píxeles) y la resolución de la misma, siendo en este caso de 1000 mm. En el ejemplo de la Figura 4.3 pueden visualizarse los parámetros que se tienen en cuenta. Al modificar las dimensiones del modelo, en milímetros, el complemento calcula automáticamente el resto de parámetros.

4.1.2. Imagen satelital

La imagen satelital se obtiene importando a QGIS las imágenes de Google. Para ello es necesario instalar otro complemento denominado *QuickMapServices*, que estará disponible en la barra de menús, **Web > QuickMapServices**, una vez instalado. Por defecto, este complemento no incluye la opción de importar imágenes de Google, por lo que en los ajustes de la misma (**Web > QuickMapServices > Settings**), se debe seleccionar *Get contributed pack* en la ventana *More services*. A continuación, para importar las imágenes satelitales hay que seleccionar **Web > QuickMapServices > Google > Google.cn Satellite**.

El siguiente paso consiste en recortar las imágenes satelitales importadas para que coincidan con la extensión delimitada por el MDT, mediante la herramienta *Convertir mapa a ráster*, disponible en la caja de herramientas de procesos. Los parámetros de esta herramienta se muestran en la Figura 4.4. Concretamente, la extensión mínima a renderizar está delimitada por la capa MDT (**... > Calcular a partir de capa**), y las unidades de mapa por píxel afectan a la resolución del ráster generado. Cuanto menor sean las unidades por píxel, mayor será la resolución de la imagen a costa de ocupar más espacio.

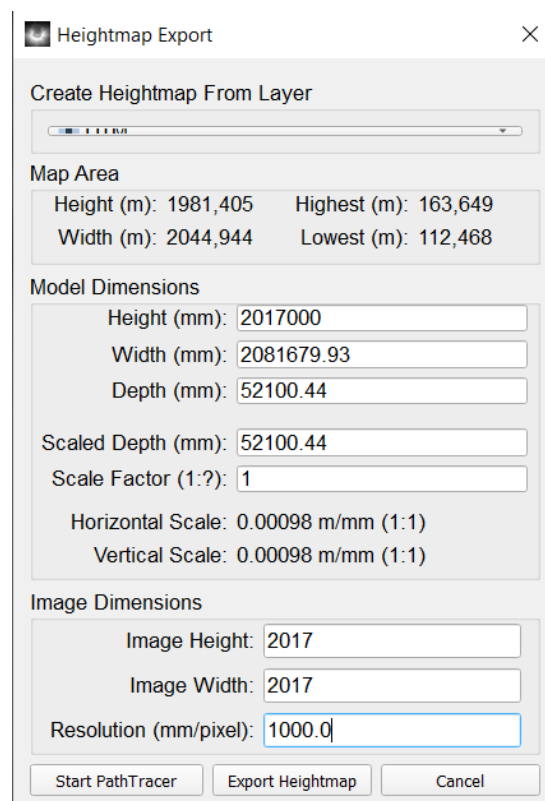


Figura 4.3. Exportar el mapa de altura

³La información sobre el mismo se encuentra en: <https://github.com/ClayJarCom/HeightmapExport>

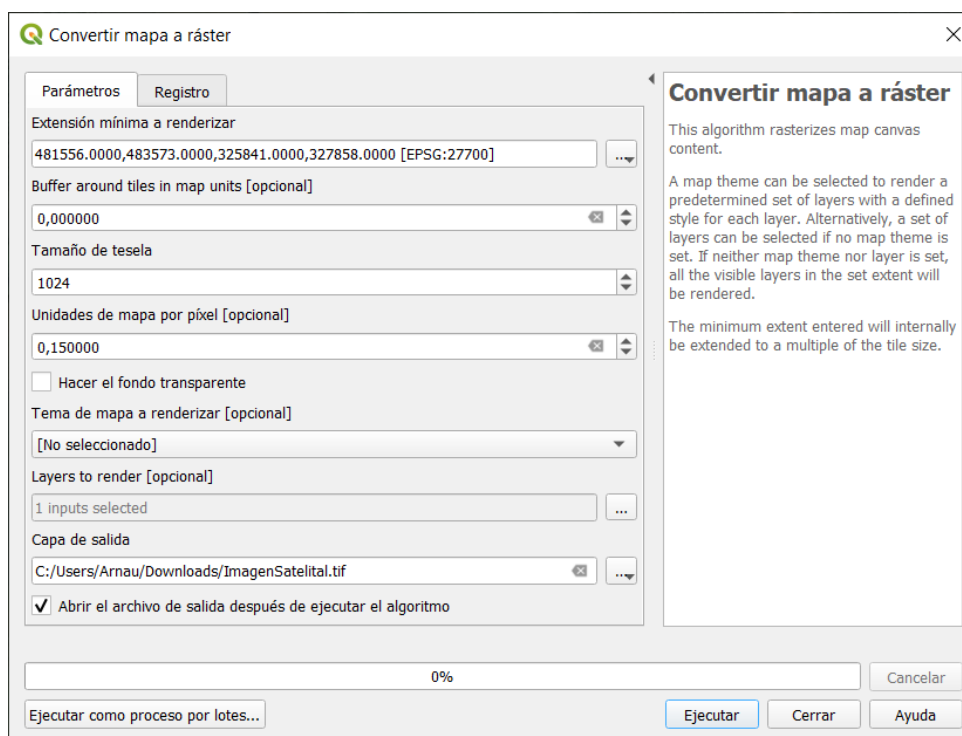


Figura 4.4. Convertir mapa a ráster

Debido a que el SRC de las imágenes satelitales importadas no corresponde con el SRC de la capa MDT, es necesario recortar el ráster generado mediante la herramienta *Cortar ráster por extensión*. En la misma se especifica el archivo de entrada (imagen satelital) y la extensión a recortar (dada por la capa MDT). Tras este proceso se obtiene la imagen satelital que servirá como textura, en formato TIFF.

Finalmente, el archivo TIFF con la imagen satelital se importa al editor de imágenes GIMP para convertirla a formato PNG. Para ello se siguen los siguientes pasos:

1. Escalar la capa al tamaño adecuado del mapa. En la información de las capas, **clic derecho sobre la capa importada > Escalar la capa**, y seleccionar el mismo tamaño del MDT, en este caso siendo de 2017 píxeles de anchura y altura.
2. Ajustar el lienzo a la nueva imagen escalada (**clic derecho sobre el lienzo > Imagen > Ajustar lienzo a las capas**).
3. Exportar a PNG, mediante la barra de menús, **Archivo > Exportar como**. En el nombre del archivo, cambiar la extensión .tif a .png, seleccionar exportar y dejar las opciones por defecto.

4.2. Crear el mapa en Unreal Engine

El mapa creado con Unreal Engine se proporciona junto al código de este proyecto, en el archivo comprimido con el nombre de *UnrealEngineMap*. Dentro de este fichero se encontrará el archivo *MathWorks_Aerospace.uproject*, el cual debe ser abierto desde Simulink, como se ha indicado anteriormente.

Una vez abierto el editor de Unreal Engine, se puede localizar el mapa por defecto en el buscador de contenidos, dentro del directorio *Maps*. Se recomienda que todos los mapas creados sean almacenados dentro de esta carpeta, en caso de que el usuario desee utilizar un mapa distinto al creado en este proyecto. Para crear el mapa, hacer clic derecho en el buscador de contenidos y seleccionar **Nivel**. Este nivel puede ser abierto haciendo doble clic sobre él.

El nuevo nivel creado no dispone de iluminación, por lo que al abrirlo se ve una pantalla en negro. Esto se puede solventar de forma rápida copiando y pegando el contenido del directorio *Lightning* que se encuentra en el mapa original (*Landscape*) en la ventana *World Outliner*, al nuevo mapa creado.

4.2.1. Importar el mapa de alturas

El siguiente paso corresponde con la creación del terreno. Para ello, en la ventana de Modos, seleccionar la opción de Terreno y dentro de la misma seleccionar *Importar desde archivo*, para indicar el archivo que contiene el mapa de altura previamente creado en QGIS. El apartado de *Material* se debe dejar como está, sin material alguno, ya que esto será modificado más adelante. Asimismo, en la resolución global se puede comprobar como el tamaño es el mismo que el especificado en QGIS a la hora de exportar el MDT.

De los parámetros modificables hay que prestar atención a la posición, rotación y escala del mapa. En general, si la resolución del mapa de altura es de 1 m por píxel, los valores de la escala X y la escala Y deberán ser iguales a 100. Por el contrario, si la resolución del mapa de altura es de 2 m por píxel, los valores de las escalas X e Y deberán ser iguales a 200, lo que equivale a multiplicar por dos. Esto se debe a que la resolución del mapa en Unreal Editor indica el tamaño del mapa (1 m/píxel), por lo que si la resolución global es de 2017x2017, el mapa será un cuadrado de 2017 m de lado, independientemente de la resolución del mapa de altura. Es por ello que se utiliza la opción de la escala, multiplicando así el tamaño del mapa para que se corresponda con la superficie del terreno real.

Por otro lado, al importar el mapa de alturas, Unreal asigna automáticamente el valor máximo de elevación a +256 m y el valor mínimo a -256 m. Para ajustar correctamente la altura del terreno, es necesario calcular el valor de la escala en Z que debe ser aplicado. Por ejemplo, si la elevación máxima del terreno es de 163.6 m y la elevación mínima es de 112.4 m, la escala Z vendrá dada por la ecuación 4.1.

$$\text{Escala Z} = \frac{163.6 - 112.4}{512} \cdot 100 = 10 \quad (4.1)$$

En cuanto a los valores de posición y orientación, estos deben ser elegidos de tal forma que el eje X del editor apunte hacia el norte del mapa, según la visualización en QGIS. En el ejemplo explicado esto se consigue estableciendo una rotación de 90° en el eje Z, y la posición X e Y igual a 100 850.0 y -100 850.0, respectivamente (las unidades en las coordenadas de la posición equivalen a centímetros). Asimismo, el centro del terreno debe coincidir con el centro del mapa, si se desea que el origen de coordenadas en el sistema de simulación corresponda con el centro del terreno.

Finalmente, en caso de que la resolución del mapa de alturas sea baja, el terreno puede poseer discontinuidades. La herramienta de suavizado, que se encuentra dentro de la opción Esculpir del

modo Terreno, como puede observarse en la Figura 4.5, permite corregir estas discontinuidades, “pintando” el mapa hasta tener una superficie sin irregularidades.

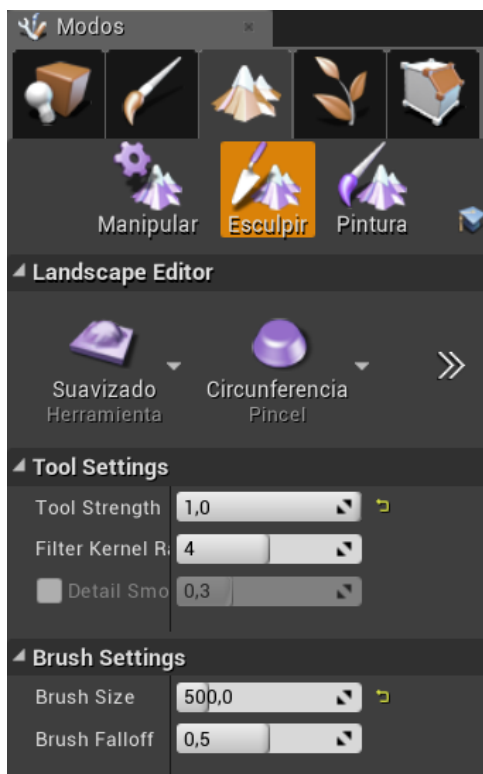


Figura 4.5. Herramienta para suavizar el terreno en Unreal Editor

4.2.2. Importar la imagen satelital como textura

La imagen satelital puede ser importada directamente al buscador de contenidos de Unreal Editor, arrastrando y soltando la imagen desde el explorador de archivos de Windows. Para crear la textura del terreno, en el directorio *Materials*, hacer clic derecho sobre el material *Landscape-FinalTexture* y seleccionar *Crear instancia del material*. Posteriormente, hacer doble clic sobre la instancia para abrir sus características y en el apartado de los parámetros activar el parámetro *Textura*, donde se tiene que seleccionar la imagen satelital importada, como se muestra en la Figura 4.6.

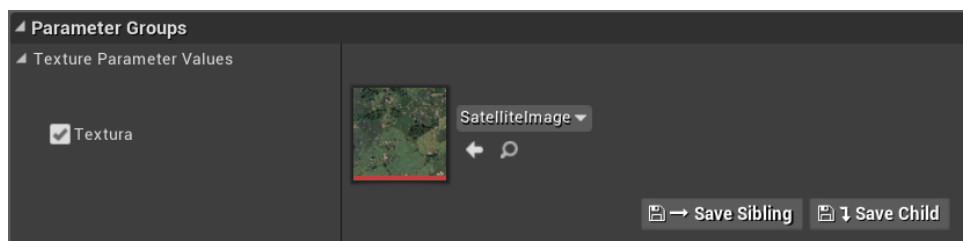


Figura 4.6. Seleccionar la imagen satelital como textura de la instancia de material

Finalmente, asignar al material del terreno la instancia creada. Para ello, seleccionar el terreno bien desde la ventana *World Outliner* o haciendo clic sobre él en la ventana central, y asignar el material en el desplegable *Landscape*, que se encuentra en la ventana de Detalles. En este punto la textura no será visible en el terreno, siendo este de color negro, ya que hay que activar

las capas del material. Esto se hace en el modo Terreno, dentro de la opción de Pintura, en el desplegable Capas. Para cada una, seleccionar el botón “+” y crear la información de la capa. Tras este proceso la textura del terreno es ya visible.

4.3. Crear las balizas terrestres

Las balizas terrestres, que deben ser identificadas por el algoritmo de visión artificial, se incluyen en el mapa creado con Unreal Editor. Estas deben ser creadas como objetos y están situadas en el directorio *Ground_Markers*. Se puede modificar el color de cada una abriendo su pestaña de edición, haciendo doble clic sobre la misma. En la pestaña abierta, la estructura de la misma puede observarse seleccionando la ventana gráfica. Para cambiar el color, seleccionar el cuadrado central (*Central_Square*) y, en el apartado de detalles, cambiar el material del mismo, como se muestra en la Figura 4.7.

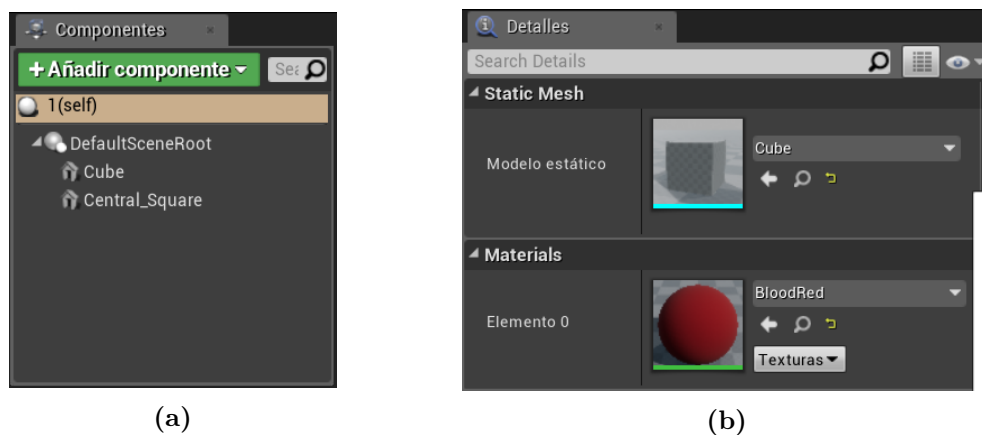


Figura 4.7. Modificar el color de las balizas

Los colores considerados en las balizas deben ser introducidos como materiales, preferiblemente dentro de la carpeta */Games/Materials/Colors*. Para ello, hacer clic derecho sobre el material denominado *BasicShapeMaterial* y seleccionar *Crear instancia del material*. Al abrir esta nueva instancia, activar el parámetro *Colour* dentro del grupo de parámetros (*Vector Parameter Values*), como se muestra en la Figura 4.8, y cambiar sus valores RGB. De esta forma pueden crearse tantos colores como se desee y, posteriormente, seleccionarlos como material del cuadrado interno de cada baliza.

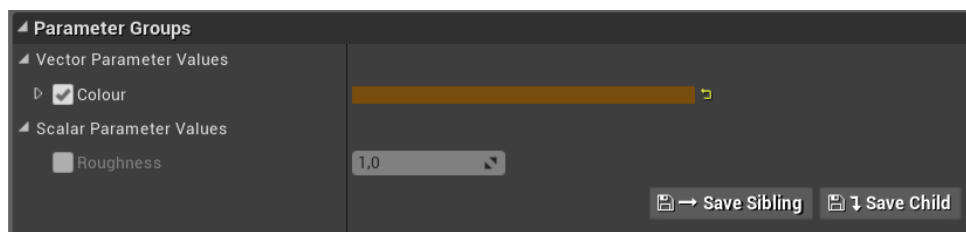


Figura 4.8. Modificar el color de las balizas

Las coordenadas de las balizas pueden obtenerse en los detalles del objeto, al hacer clic sobre las mismas. Cabe destacar que estas coordenadas corresponden con una esquina del objeto, y no de su punto central. Para obtener este punto central se recomienda insertar un plano en el mapa, que por defecto es un cuadrado de 1 m de lado. Después, desplazar y rotar el plano para

que coincida con el cuadrado central de color de la baliza. Para alternar entre las acciones de rotar y desplazar el objeto, habiéndolo seleccionado, presionar la tecla E o W, respectivamente. Finalmente, en la información sobre el plano insertado se tendrán las coordenadas de su punto central en centímetros respecto del origen de coordenadas, que coincidirán con el punto central del objeto. Cabe destacar que la obtención de las coordenadas no es necesaria para aquellas que sean utilizadas en el entrenamiento o la evaluación del detector, ya que la posición solo se utiliza para la misión final.

4.4. Empaquetar el proyecto

El empaquetado del proyecto se realiza para generar el ejecutable que más tarde será utilizado por los bloques de simulación en Simulink. Para ello, se recomienda comprobar que en **Editar** > **Ajustes del proyecto** > **Empaquetado** estén los siguientes parámetros⁴:

- Usar archivo Pak → Activado.
- *Cook everything in the project content directory (ignore list of maps below)* → Desactivado.
- *Cook only maps (this only affects cookall)* → Activado.
- *Create compressed cooked packages* → Activado.
- *Exclude editor content when cooking* → Activado.
- *List of maps to include in a packaged build* → mapa que se quiere empaquetar para utilizar en la simulación (i.e. /Games/Maps/Landscape).
- Directorios de archivos adicionales para procesar → tienen que estar definidos los siguientes directorios, que se pueden añadir mediante el símbolo +:
 - /MathWorksSimulation/Characters
 - /MathWorksSimulation/Weather
 - /MathWorksSimulation/Vehicles
 - /MathWorksSimulation/VehiclesCommon
 - /MathWorksUAV

Es importante realizar otro paso previo antes de empaquetar el proyecto. En la barra de herramientas, seleccionar **Blueprints** > **Open Level Blueprint**. A continuación, seleccionar **Archivo** > **Reparent Blueprint** > **Sim3dLevelScriptActor** en la ventana abierta, como se muestra en la Figura 4.9. Antes de cerrar la ventana es importante guardar los cambios. Saltarse algún paso en este proceso puede producir el error dado en la Figura 4.10, cuando se intenta ejecutar el archivo de Simulink.

Finalmente, para empaquetar el proyecto seleccionar **Archivo** > **Empaquetar el proyecto** > **Windows** > **Windows (64-bit)** y elegir el directorio en el cual se encuentra el proyecto. El resultado del empaquetado se almacenará en un directorio con nombre *WindowsNoEditor*, dentro del cual estará la aplicación con extensión **.exe**. La simulación a partir de este ejecutable se especifica en el bloque *Simulation 3D Scene Configuration*, donde se elige la opción de *Unreal Executable* dentro de *Scene source*. En el nombre del archivo hay que proporcionar la ruta del archivo **.exe** generado, así como el mapa que se quiere utilizar (i.e. /Games/Maps/Landscape).

⁴Si se utiliza el editor con español como lenguaje, no se traducen todos títulos, por lo que se explican según aparecen con el fin de evitar confusiones.

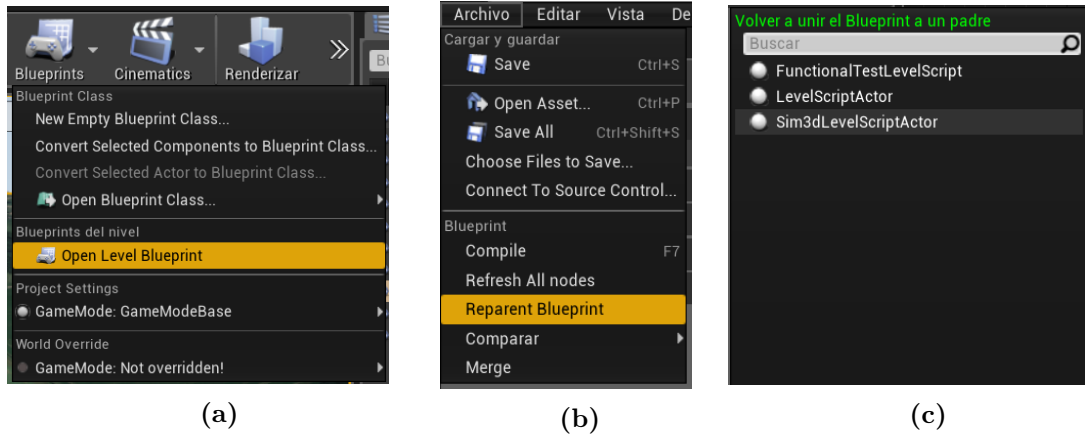


Figura 4.9. Paso previo al empaquetado

```

MATLAB System block 'HORUS\_control/Animación/Tipo de simulación/Simulación realista/Simulation 3D Scene Configuration/Simulation 3D Engine' error occurred when invoking 'resetImpl' method of 'Simulation3DEngine'. The error was thrown from '
'C:\Program Files\MATLAB\R2020b\toolbox\shared\sim3d\sim3d\+sim3d\EngineInterface.p' at line 0
'C:\Program Files\MATLAB\R2020b\toolbox\shared\sim3d\sim3d\+sim3d\Engine.p' at line 0
'C:\Program Files\MATLAB\R2020b\toolbox\shared\sim3d\sim3d\+sim3d\Project.p' at line 0
'C:\Program Files\MATLAB\R2020b\toolbox\shared\sim3dblks\sim3dblks\Simulation3DEngine.p' at line 0'.
Caused by:
  • 3D simulation engine interface setup error: could not establish communication. Please restart the simulation.
Component: Simulink | Category: Block error

```

Figura 4.10. Error en la ejecución de Simulink debido a un mal empaquetado del escenario

5 | Visión artificial

El presente capítulo sirve como guía para utilizar los algoritmos de visión artificial creados en el proyecto. Además, se explica el proceso de entrenamiento y evaluación del mismo, para aquellos usuarios que deseen entrenar su propio modelo a partir de datos distintos. Del mismo modo, se detalla el proceso de cálculo de las coordenadas de las balizas y el análisis de los resultados. Estos algoritmos se implementan en el subsistema **Visión artificial** de la Figura 1.2, cuyo contenido se muestra en la Figura 5.1.

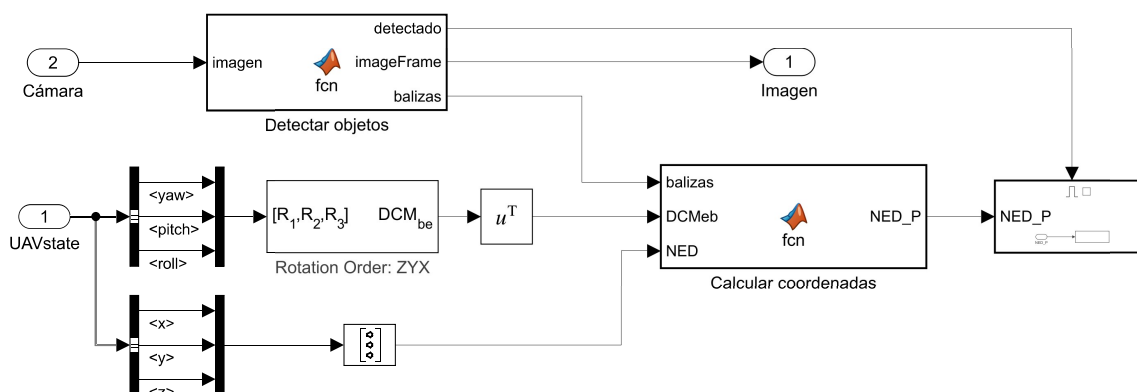


Figura 5.1. Implementación en *Simulink* de la detección de balizas

5.1. Detección de objetos

El algoritmo de visión artificial debe ser entrenado y evaluado a partir de los datos generados en la simulación. Como primer paso, deben definirse dos misiones distintas en QGC, cada una para conseguir el vídeo con los datos necesarios. La planificación de las mismas debe hacerse teniendo en cuenta la posición de las balizas en el mapa, las cuales se tienen que agrupar en dos conjuntos, uno para el entrenamiento y el otro para la evaluación del algoritmo. En el código proporcionado junto a este proyecto pueden encontrarse estas misiones definidas en el directorio **QGC/**.

Durante la simulación, en el subsistema **HORUS_control > Animación** se encuentra un interruptor para empezar a grabar. Este interruptor modifica la variable que permite activar o desactivar el subsistema *Obtener vídeo*, dentro del cual se encuentra el bloque *To Multimedia File*, que guarda las imágenes producidas durante la simulación en el archivo indicado. Según el vídeo que se quiera obtener, debe cambiarse el nombre del archivo especificado en este bloque así como la ruta donde se quiere almacenar. Además, se recomienda empezar a grabar en el

momento previo en que la primera baliza sea visible, y dejar de grabar tras visualizar la última baliza, reduciendo así la duración total del vídeo y por tanto el espacio que ocupa en memoria.

5.1.1. Entrenamiento

El entrenamiento se hace a partir del vídeo obtenido en la simulación, como se acaba de explicar. El primer paso consiste en generar los datos de verdad terreno, lo cual se hace utilizando la aplicación *Video Labeler*, que puede abrirse escribiendo *videoLabeler* en la ventana de comandos o desde la pestaña de **APPS > IMAGE PROCESSING AND COMPUTER VISION > Video Labeler**.

Tras abrir la aplicación, cargar el vídeo de entrenamiento desde el desplegable **Load > Video**. Después de cargarlo se puede reproducir o avanzar un fotograma a la vez en la barra de opciones inferior. A continuación, definir una nueva etiqueta para las balizas, desde la pestaña **ROI Labels > Define new ROI label**, como se muestra en la Figura 5.2. Con esto, al aparecer la primera baliza en el vídeo, crear un cuadrado haciendo clic sobre una esquina de la misma y arrastrando el cursor hasta la esquina opuesta.

Una vez se ha definido el cuadrado para la primera baliza, seleccionar el desplegable **Select Algorithm > Point Tracker**. Este algoritmo automatiza el proceso de identificar las regiones de interés en cada fotograma, reduciendo significativamente el tiempo necesario para definir todos los datos. Los algoritmos pueden ser creados por el usuario, según sus necesidades, o pueden utilizarse los que vienen creados por defecto. El algoritmo *Point Tracker* que se encuentra por defecto es más que suficiente para identificar las balizas del vídeo, por lo que no es necesario destinar tiempo a crear algoritmos propios. Por último, estando la baliza identificada y el algoritmo seleccionado, hacer clic en **Automate**.

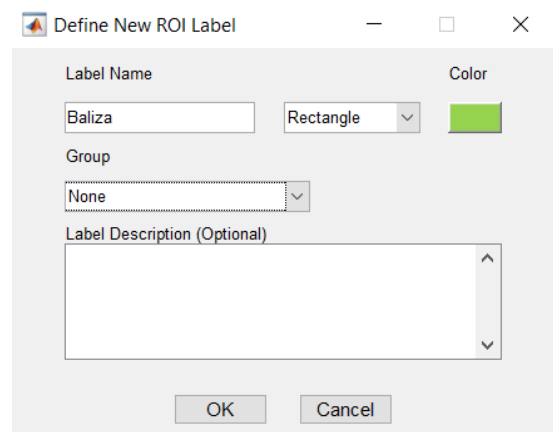


Figura 5.2. Definir la región de interés (ROI)

En la pestaña abierta, basta con hacer clic en *Run* para que el algoritmo detecte automáticamente la baliza en cada fotograma. Una vez finaliza, hacer clic en *Accept*. Después de este procedimiento se puede comprobar como la baliza seleccionada inicialmente se encuentra delimitada por la región de interés en cada fotograma. Sin embargo, este no es el caso de las subsecuentes balizas que aparecen en el vídeo, por lo que hay que repetir el proceso descrito para cada una de las balizas. Al finalizar, las etiquetas deben ser exportadas, preferiblemente a un archivo, mediante el desplegable **Export Labels > To File**, lo cual guardará los datos en el archivo *.mat* que se indice. Adicionalmente, se recomienda guardar la sesión en otro archivo *.mat*, en caso de que se desee modificar algún dato de la misma en un futuro.

Dentro del código de este proyecto se encuentra el directorio **VisionArtificial/**, que contiene las sesiones para la identificación de las balizas, con el identificativo *videoLabelingSession-**, tanto del vídeo de entrenamiento como en el vídeo de evaluación. Además, se incluyen los datos de verdad terreno exportados de la aplicación *Video Labeler*, bajo el nombre de *GTruthTraining.mat*.

Finalmente, el entrenamiento del detector se realiza en el *script* *AlgoritmoVA.m*, dentro del directorio **VisionArtificial/**. Dentro de este se cargan los datos de verdad terreno, se selecciona la etiqueta con la cual se han identificado las balizas (i.e. “Baliza”), se entrena el modelo y se almacena el resultado en el directorio **ArchivosDeDatos**. Aspectos importantes a tener en cuenta de este *script* son:

- El directorio **VisionArtificial/DatosEntrenamiento** debe estar vacío antes de ejecutar el *script*.
- En la línea 27, la función *objectDetectorTrainingData* acepta como parámetro el factor de muestreo, que indica cuántas imágenes escribe en el directorio de destino. Por ejemplo, si el parámetro es igual a 4, la función escribirá una imagen en el directorio de destino cada 4 imágenes que dispone de los datos de verdad terreno.
- En la línea 31 se utiliza la función *trainACFObjectDetector* para entrenar el detector, con los parámetros explicados en la memoria del proyecto. Esta función puede ser reemplazada por otro tipo de detectores que empleen distintos algoritmos, según las necesidades del usuario¹.

Siguiendo los pasos explicados en este apartado, el usuario tiene la opción de entrenar un detector acorde a sus necesidades, obteniendo primero los datos de verdad terreno o utilizando los proporcionados en este proyecto, para posteriormente entrenarlo según el *script* proporcionado.

5.1.2. Evaluación

El detector entrenado puede ser evaluado mediante los archivos *PruebaDetector.m* y *EvaluacionDetector.m*, ambos en el directorio **VisionArtificial/**. El primero puede ser utilizado para visualizar la detección de las balizas en el segundo vídeo que debe obtenerse, correspondiente a los datos de evaluación. Por otro lado, el segundo se utiliza para obtener las métricas de evaluación del detector, explicadas en detalle en la memoria del proyecto.

5.1.3. Implementación en *Simulink*

La implementación del detector de balizas se incluye en la función *Detectar objetos* dentro del subsistema **HORUS_control > Animación > Visión Artificial > Con detección de objetos**, que puede ser abierta haciendo doble clic en ella.

Debido a la estructura de las señales y las variables que utiliza Simulink, no es posible pasar de forma íntegra el detector entrenado a esta función. Este detector está almacenado en una estructura en el archivo *Detector.mat*, en el cual los campos *Classifier* y *TrainingOptions* almacenan toda la información relevante para reconstruir el detector. El segundo campo mencionado contiene variables de tipo *char*, las cuales producen errores al importar estos campos como parámetros en la función *Detectar objetos*, por lo que son eliminadas en la sección *Datos del detector* dentro del archivo *MasterScript.m*.

Si bien el código proporcionado no genera errores, al crear un nuevo detector los campos eliminados pueden cambiar, por lo que se recomienda verificarlos mediante los siguientes pasos:

¹La información sobre los tipos de detectores que pueden ser utilizados se encuentra en: <https://es.mathworks.com/help/vision/object-detection.html>

1. En la ventana de comandos de MATLAB, comprobar los valores de los campos eliminados introduciendo ²:


```
>>detectorStruct.TrainingOptions.ModelName
>>detectorStruct.TrainingOptions.hog.Interpolation
```
2. Dentro de la función *Detectar objetos* de Simulink, comprobar que los valores asignados son iguales que los del paso previo (líneas 14 y 15 del código).

Por otro lado, modificando la sentencia condicional dentro del código de esta función (línea 19), se puede modificar los fotogramas por segundo analizados. Por ejemplo, si la condición de la sentencia condicional corresponde a **iteration == 20**, el detector analizará únicamente 1 de cada 20 imágenes producidas por la cámara dentro del entorno de visualización. A su vez, si la cámara incluida en el entorno de visualización produce 20 fotogramas por segundo (definido en la línea 63 del archivo *MasterScript.m*), entonces el detector solo analizará un fotograma por segundo.

Finalmente, si las balizas se han dispuesto en el mapa de forma que aparezcan varias en un mismo fotograma dentro de la simulación, es recomendable modificar el número máximo de balizas consideradas por esta función (línea 5 del código de la función). Por defecto está definido en 5 detecciones máximas por fotograma, por lo que si el detector devuelve un mayor número de detecciones se producirá un error. En caso de que esto suceda, basta con aumentar el valor de detecciones máximas.

5.2. Cálculo de las coordenadas

El procedimiento para el cálculo de las coordenadas de las balizas, explicado con detalle en la memoria del proyecto, se implementa en la función *Calcular coordenadas*, disponible en el mismo subsistema que las funciones de detección de objetos y texto.

En esta función el usuario debe definir los parámetros ópticos y la posición de la cámara en la primera sección del código. Cabe destacar que en este caso la posición de la cámara debe estar dada en ejes cuerpo (eje Z hacia abajo), mientras que en el bloque de la cámara (disponible en **HORUS_control > Animación > Tipo de simulación > Simulación realista**), el sentido positivo del eje Z es hacia arriba. Esto es, si en el bloque de la cámara la traslación se define como [0 0 -0.15], en la función debe estar definido como [0 0 0.15]. Por el contrario, los ejes X e Y se mantienen iguales.

Por otro lado, si el usuario así lo considera, puede modificar las siguientes variables:

- *iteracionesBucle* (línea 15): como se indica en la memoria del proyecto, la posición de la baliza se calcula escalando un vector normalizado y comparando la altura del mismo en un punto dado con la altura proporcionada por el Modelo Digital del Terreno. Esto se realiza en un bucle que se repite el número de veces indicado por esta variable. Por ejemplo, si la variable tiene un valor de 150, en la última iteración el vector original que indica la dirección en la que está la baliza será multiplicado por 150.

En caso de que la aeronave esté volando a una altura elevada, este valor puede ser insuficiente para alcanzar la altura del terreno, lo cual sería visible fácilmente en los resultados de

²Como consejo, después de escribir el punto, si se presiona el tabulador del teclado aparecen los campos de la estructura.

la simulación. Si los resultados muestran un error en el eje z elevado, esta variable deberá ser aumentada.

- *NED_P* (línea 16): esta variable inicializa la matriz donde se almacenan las posiciones de las balizas detectadas. El número de filas de esta variable indica el valor máximo de balizas por fotograma considerado. Si este valor ha sido modificado en la función *Detectar objetos*, también deberá ser modificado en esta función.

Finalmente, la posición de la baliza se determina según el error mínimo entre la altura dada por el vector y la altura del terreno real. Por ello, un parámetro de entrada de esta función corresponde con la variable MDT, almacenada en **ArchivosDeDatos/MDT.mat**. Para generar esta variable a partir del mapa de altura se proporciona el archivo *DatosMDT.m*, disponible en el directorio **QGIS/**. En este debe especificarse la elevación máxima y mínima del terreno, así como las dimensiones del mapa de altura.

5.3. Análisis de resultados

Tanto las posiciones de las balizas como el texto identificado en cada una se envían al **workspace** de MATLAB una vez finalizada la simulación, dentro de la estructura definida por la variable *out*. El análisis de estas variables se realiza en el archivo *AnalisisResultados.m* proporcionado en el directorio **AnalisisResultados/**.

El error en la estimación de la posición de las balizas se calcula comparando los valores estimados con los reales. Los valores reales están definidos en el archivo *posicionRealBalizas.mat*, dentro del directorio **ArchivosDeDatos/**. Para generar estos datos, se proporciona el archivo *CoordenadasBalizas.m*, dentro del mismo directorio en el que se encuentra el archivo para analizar los resultados, para definir las posiciones de las balizas en el escenario. En este archivo se crea una estructura de datos con dos campos, uno indicando la letra y el otro la posición de la baliza en coordenadas NED. La posición de las balizas es la obtenida en Unreal Editor, como se explica en la sección 4.3.

