Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio

Grado en Ingeniería Aeroespacial

Trabajo de Fin de Grado

Diseño de sistemas de control adaptativo para el vuelo mediante redes neuronales

Autor: Ignacio Vidal de Arce
Especialidad: Ciencias y Tecnologías Aeroespaciales
Tutor del trabajo: Ignacio Gómez Pérez
Cotutor: Borja Gómez López

15 de junio de 2023

Resumen

El objetivo de este Trabajo de Fin de Grado es estudiar la posibilidad de utilizar la Inteligencia Artificial y el Deep Learning para mejorar el comportamiento de un avión en vuelo. Para ello, se toma como ejemplo la dinámica longitudinal de un UAV con configuración de avión de ala fija y se acude a las técnicas de control moderno (MIMO) para diseñar controladores en bucle cerrado y servomecanismos de seguimiento (de la mano de Matlab y Simulink).

A partir de dichas técnicas, se entrenan redes neuronales con librerías de Python para obtener un controlador autoadaptativo que proporcione las ganancias óptimas en cada punto a lo largo de toda la envolvente de vuelo. Se comparan los resultados con los de la planificación de ganancias (*gain scheduling*). Además, se amplía el modelo incluyendo variaciones en la configuración másica de la aeronave para permitir al controlador alcanzar situaciones óptimas en un mayor rango de puntos de operación.

Se encuentra que con el diseño de estas redes se consiguen mejoras frente a la planificación de ganancias, que se trata de un método relativamente sencillo y que funciona adecuadamente y que no son necesarios demasiados puntos de entrenamiento para alcanzar resultados satisfactorios.

Agradecimientos

La vida, según parece, es un camino de escalones que hay que ir superando, y la etapa que acaba con este Trabajo de Fin de Grado sin duda ha sido la mayor de mis escaleras hasta el momento.

En primer lugar, quería dar las gracias a toda mi familia. En especial, a mis padres, mi hermana, mis abuelas y mi abuelo, que algo sabe de volar. Habéis sido un pilar inamovible e indispensable y sin vosotros hubiera sido muy distinto. No me puedo olvidar de mi círculo de amistades, todos sabéis quiénes sois. Muchas gracias también, Marina. Va dedicado a vosotros.

También querría agradecer el trabajo de los profesores que consiguen despertar tus inquietudes, alimentar tu curiosidad y empujarte a seguir adelante. Sin duda, estos magníficos docentes, por desgracia no lo habitual, han contribuido de manera más que enriquecedora a mi formación y han suavizado la experiencia global mi paso por la universidad. Esto se extiende a todos mis compañeros y amigos, fieles aliados en todo este camino.

Índice general

Abstracto III

Agradecimientos V

Índice de figuras XIII

Índice de tablas XV

Índice de listings XVII

1. Introducción 1

1.1. Proyecto: *qué* . . . 1

1.2. Motivación: *por qué* . . . 2

1.3. Metodología: *cómo* . . . 3

2. Modelo de UAV 5

2.1. Introducción . . . 5

2.2. Modelo dinámico general . . . 5

2.3. Modelo dinámico longitudinal . . . 7

2.3.1. Sistema de ecuaciones . . . 8

2.3.2. Fuerzas y momentos aerodinámicos . . . 8

2.3.3. Fuerza de empuje . . . 10

2.3.4. Modificaciones por centro de gravedad variable . . . 10

2.4. Parámetros del modelo de UAV . . . 11

2.5. Modelo en Matlab . . . 12

2.6. Envolvente de vuelo . . . 13

3. Técnicas de control automático **19**

3.1. Introducción . . . 19

3.2. Representación en el espacio de los estados . . . 19

3.3. Trimado y linealización . . . 20

3.4. Respuesta en bucle abierto . . . 21

3.5. Diseño del controlador . . . 23

3.5.1. Elección de las matrices de costes del controlador . . . 24

3.6. Diseño del sistema de tracking . . . 27

3.6.1. Caso de seguimiento de una única variable . . . 27

3.6.2. Caso de seguimiento de varias variables . . . 28

3.6.3. Obtención de las matrices de costes para cada sistema de tracking . . . 28

4. Diseño del controlador mediante una red neuronal **33**

4.1. Introducción . . . 33

4.2. Evaluación del Índice de Desempeño . . . 34

4.3. Diseño y entrenamiento de la red neuronal . . . 39

4.3.1. Generación del dataset . . . 39

4.3.2. Diseño y arquitectura . . . 40

4.4. Comportamiento del controlador frente a una perturbación . . . 41

4.5. Comportamiento del controlador en un sistema de tracking . . . 43

4.6. Consideraciones sobre el diseño de la red neuronal . . . 48

4.6.1. Arquitectura y dataset . . . 49

4.6.2. Ajuste de hiperparámetros: *training batch size* y *drop probability* . . . 51

4.6.3. Función de activación . . . 54

4.6.4. *Learning rate* . . . 56

5. Ampliación del modelo a más dimensiones y mejoras generales **59**

5.1. Introducción . . . 59

5.2. Ampliación del dataset a cinco dimensiones . . . 60

5.3. Resultados de las redes neuronales en cinco dimensiones . . . 62

5.4. Análisis de los efectos de la simplificación del dataset . . . 64

5.5. Análisis de las implicaciones de una situación real . . . 67

6. Conclusiones **71**

6.1. Resumen de los principales resultados 71

6.2. Consideraciones para trabajos futuros 72

6.3. Nota del autor 75

A. Códigos de Matlab **77**

A.1. Modelo no lineal de UAV 77

A.2. Trimado, linealización y diseño del controlador 80

A.3. Sistema de tracking 81

A.4. Generación del dataset 83

A.5. Funciones del los subsistemas del controlador discreto y de la red neuronal . 85

B. Códigos de Python **87**

B.1. Creación, entrenamiento y evaluación de la red neuronal 87

B.2. Función para la inferencia del controlador con la red neuronal desde Matlab 93

C. Modelos de Simulink **97**

C.1. Modelo para trimar el UAV no lineal 97

C.2. Modelo de UAV lineal en bucle abierto 98

C.3. Modelo de UAV lineal en bucle cerrado 99

C.4. Modelo de un sistema de tracking con el UAV no lineal 100

C.5. Modelo con la combinación de todos los sistemas de tracking para el UAV no lineal: controlador discreto, gain scheduling y red neuronal 101

Índice de figuras

1.1. Modelo de UAV de referencia. 2

2.1. Perfil aerodinámico NACA 2410 13
2.2. Curvas del coeficiente de sustentación del NACA 2410. 14
2.3. Ángulo de ataque para vuelo rectilíneo horizontal en función de la velocidad y para distintas alturas. 15
2.4. Deflexión del timón de profundidad para vuelo equilibrado. 15
2.5. Fronteras de la envolvente de vuelo a bajas velocidades. 16
2.6. Fuerzas de resistencia y empuje en función de la velocidad y para distintas alturas. 17
2.7. Frontera de la envolvente de vuelo a altas velocidades. 17
2.8. Envolvente de vuelo del UAV. 18

3.1. Respuesta en bucle abierto a una perturbación de velocidad. 22
3.2. Respuesta de la altura en bucle abierto a una perturbación de velocidad. . . 23
3.3. Respuesta de la velocidad en bucle abierto a un escalón en el timón de profundidad. 23
3.4. Respuesta en bucle cerrado con el primer diseño del controlador. 26
3.5. Respuesta en bucle cerrado con el segundo diseño del controlador. 26
3.6. Respuesta a un sistema de tracking para la velocidad siguiendo una onda cuadrada. 29
3.7. Respuesta a un sistema de tracking de velocidad y altura siguiendo dos rampas. 30

4.1. Respuesta en un punto de operación de baja velocidad con dos controladores distintos. 35

4.2. Respuesta en un punto de operación de alta velocidad con dos controladores distintos. 36
4.3. Diagrama de flujo para la evaluación del PI con un controlador de referencia. 37
4.4. Índice de Desempeño de dos controladores a lo largo de la envolvente de vuelo. 37
4.5. Diagrama de flujo para la evaluación del PI con un controlador diseñado en cada punto. 38
4.6. Índice de Desempeño de un controlador diseñado en cada punto de operación a lo largo de toda la envolvente de vuelo. 38
4.7. Procedimiento 40
4.8. Esquema de la arquitectura de la red neuronal. 41
4.9. Comparación de la respuesta frente a una perturbación entre un controlador original y la red red neuronal. 42
4.10. Comparación de los Índices de Desempeño entre un controlador diseñado en cada punto de operación y la red neuronal. 42
4.11. Comparación de la respuesta frente a una perturbación entre el controlador original y la red neuronal. 43
4.12. Respuesta a un sistema de tracking de velocidad y altura con un único controlador. 44
4.13. Respuesta a un sistema de tracking de velocidad y altura con una familia de controladores definidos cada 5 m/s. 45
4.14. Respuesta a un sistema de tracking de velocidad y altura con un sistema de planificación de ganancias. 46
4.15. Respuesta a un sistema de tracking de velocidad y altura con el controlador que proporciona la red neuronal. 47
4.16. Comparación del Índice de Desempeño entre controlador original y la red neuronal para distintas arquitecturas. 50
4.17. Funciones de activación de la red neuronal 55

5.1. Respuesta a un sistema de tracking de velocidad y altura con el controlador que proporciona una red neuronal *pobre*. 63
5.2. Ganancias del controlador para distintas alturas y en función de la velocidad. 67

5.3. Respuesta a un sistema de tracking de velocidad y altura con un sistema de planificación de ganancias reducido *real*. 69
5.4. Respuesta a un sistema de tracking de velocidad y altura con el controlador que proporciona una red neuronal con un dataset reducido *real*. 70

C.1. `UAVTrimh.slx`: Modelo no lineal para el trimado y linealización. 97
C.2. `UAVLinOL_step.slx`: Modelo lineal en bucle abierto con entrada escalón. . . 98
C.3. `UAVLinCL.slx`: Modelo lineal en bucle abierto con entrada escalón. 99
C.4. `UAVNLCL_TRACK.slx`: Modelo para el sistema de tracking. 100
C.5. `UAV_TRACK.slx`: Modelo completo de seguimiento. 101
C.6. `UAV_TRACK\sub1.slx`: Subsistema para el controlador discreto. 102
C.7. `UAV_TRACK\sub2.slx`: Subsistema para el Gain Scheduling. 102

Índice de cuadros

2.1. Parámetros geométricos y másicos. 12
2.2. Parámetros y coeficientes del grupo motopropulsor. 12
2.3. Parámetros y coeficientes aerodinámicos. 12

4.1. Comparación de la bondad de las distintas redes y datasets. 51
4.2. Hiperparámetros con el dataset A. 53
4.3. Hiperparámetros con el dataset B. 53
4.4. Variación del ritmo de aprendizaje. 56

5.1. Comparación de las distintas arquitecturas de red y su probabilidad de drop. 62
5.2. Comparación de los resultados de varias arquitecturas de red según el dataset utilizado. 65

Índice de listings

A.1. `UAVNL.m`: Función del modelo no lineal del UAV. 77
A.2. `trimlinlqr.m`: Trimado, linealización y diseño del controlador. 80
A.3. `tracking.m`: Diseño del sistema de tracking. 81
A.4. `GenerateDataset.m`: Generación del dataset. 83
A.5. `state2k_discrete.m`: Función del controlador discreto. 85
A.6. `state2k.m`: Función del controlador con la red neuronal. 86
B.1. `NNScript.py`: Entrenamiento de la red neuronal. 87
B.2. `state2k.py`: Función para inferir el controlador con la red neuronal. 93

Capítulo 1

Introducción

El mundo de la aeronáutica lleva evolucionando desde su creación, explorando las tecnologías disponibles en cada momento y siendo uno de los mayores focos de desarrollo científico y tecnológico de nuestra sociedad.

Motivado por ese marco de trabajo, espíritu de mejora e innovación, este Trabajo de Fin de Grado de Grado, de la titulación de Grado en Ingeniería Aeroespacial en la Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio de la Universidad Politécnica de Madrid, trata de aportar un pequeño estudio al gran ecosistema en el que vive la Ingeniería Aeronáutica.

A continuación se dedican unos apartados a intentar responder a las preguntas de: *qué* se ha hecho en este trabajo, *por qué* se ha elegido este estudio en particular y cómo se ha llevado a cabo.

1.1. Proyecto: *qué*

En este Trabajo de Fin de Grado se ha desarrollado un método de diseño de un controlador autoadaptativo para el control del vuelo de un avión mediante Inteligencia Artificial.

Para ello se ha obtenido un modelo sencillo aunque con suficiente complejidad de un UAV radiocontrol con configuración de avión de ala fija partiendo de las ecuaciones de la Mecánica del Vuelo. Una referencia gráfica se encuentra en la figura 1.1. A partir del modelo se han desarrollado distintas técnicas de control empleando la teoría de control moderno MIMO (*Multiple Input Multiple Output*), que ha resultado ser muy útil para el análisis y el diseño de controladores para este tipo de sistemas. Estos sistemas de control serán la referencia a partir de la que se diseñará el controlador adaptativo. Esta tarea se ha abordado con la creación de redes neuronales en el marco del *Deep Learning*, una rama del *Machine Learning* que, a su vez, existe dentro de la Inteligencia Artificial.

Así, por medio de técnicas del control MIMO (como el LQR, *Linear Quadratic Regau-*

Figura 1.1: Modelo de UAV de referencia: *Skywalker*[1].

lator) se han diseñado diferentes controladores «óptimos» en distintos puntos de operación de la envolvente de vuelo del UAV. Se ha desarrollado también un sistema de seguimiento para poder simular el desplazamiento del punto de operación del UAV dentro de dicha envolvente.

El objetivo, por tanto, consiste en utilizar dichos controladores óptimos en determinados puntos para entrenar una red neuronal que sea capaz de proporcionar las ganancias más apropiadas en cada momento según el estado en el que se encuentre el UAV. Este método se comparará con la planificación de ganancias (*gain scheduling*), que es un método más conocido y muy utilizado en aeronaves reales.

1.2. Motivación: *por qué*

El interés de realizar un estudio como el presente puede estar impulsado por muchos motivos. En primer lugar, el objetivo es tratar de encontrar una forma de volar en condiciones óptimas, lo que siempre lleva asociado un aumento de las eficiencias, y esto en la industria puede traducirse en una disminución de costes y un aumento de las rentabilidades. Además, otro punto a favor es todo lo relacionado con la disminución de el impacto medioambiental que tiene la actividad aérea y con el desarrollo de la sostenibilidad del sector.

De la misma forma, el análisis recogido en estas páginas logra combinar las técnicas modernas de control con las *aun más modernas* capacidades de la Inteligencia Artificial. Este estudio logra aprovechar una muy pequeña fracción de la potencia y prestaciones que se pueden alcanzar con las redes neuronales, sirviendo para encontrar una nueva aplicación a este prometedor tipo de tecnologías.

Por otro lado, otro motivo para la realización de este estudio reside en un interés

[1] Obtenido de: https://hobbyking.com/es_es/skywalker-1680-v6-fpv-platform-epo-kit.html.

personal de seguir profundizando en las técnicas de control más allá de lo tratado en el grado. Asimismo, este proyecto me ha brindado la oportunidad de adentrarme en el mundo de la Inteligencia Artificial, combinándolo de manera muy enriquecedora con el Control Automático y la Mecánica del Vuelo.

1.3. Metodología: *cómo*

El desarrollo temporal del proyecto ha tenido distintas etapas. Al comienzo, el primer foco de estudio fue la teoría detrás de las redes neuronales y sobre su implementación en Python para su uso. Paralelamente, una vez caracterizada la aeronave, tuve que profundizar en las técnicas de control automático (como la teoría detrás del LQR y el funcionamiento de los servomecanismos de tracking) para poder definir el problema que trataba de resolver. El diseño de todos estos sistemas se llevó a cabo combinando funciones del paquete *Control System Toolbox* de Matlab y los métodos de simulación de Simulink. El sustento teórico detrás de estos sistemas de control se basa en Stevens et al. (2015) y Williams et al. (2007). Por otro lado, el desarrollo de las redes neuronales empleadas para el Deep Learning se hizo empleando librerías de Python, como PyTorch y ScikitLearn, cuya documentación también puede encontrarse en las referencias de este texto: Paszke et al. (2019) y Pedregosa et al. (2011).

A continuación se detalla con más detalle el orden y los contenidos del trabajo.

En primer lugar, la deducción del modelo dinámico del UAV, los valores numéricos de todos los parámetros implicados y la obtención de su envolvente de vuelo se encuentra en el capítulo 2. Este contiene el fundamento de la dinámica que hay detrás de este estudio y recoge los principales aspectos relacionados con la Mecánica del Vuelo del UAV.

Por otro lado, el diseño de los mencionados sistemas de control se cubre en el capítulo 3. En él, en primer lugar, se fija la representación del sistema en el espacio de los estados y se presentan los métodos para trimar y linealizar el sistema. Además, se explora brevemente su respuesta en bucle abierto. Considerando ya el primer sistema de control por realimentación en bucle cerrado, se repasa la teoría del LQR y se discuten los valores más apropiados para la elección de las matrices de costes asociadas. Por último, se resume el procedimiento para el diseño de un sistema de tracking, tanto de una variable como de dos, y se detallan las particularidades de cómo obtener respuestas satisfactorias.

Obtenidos dichos sistemas de control, el capítulo 4 comienza definiendo una forma de evaluar el desempeño de un controlador en cualquier punto de operación. El resto del capítulo se dedica a diseñar la red neuronal que estará detrás del controlador adaptativo. Se describe la forma de obtener los puntos de entrenamiento, se analiza la estructura de la red, y se obtienen los primeros resultados. Posteriormente se compara el comportamiento que proporciona la red neuronal frente a un sistema de seguimiento con otras técnicas (como el mencionado *gain scheduling*). El capítulo acaba dedicando unas páginas a la influencia que tienen diversos hiperparámetros del proceso de entrenamiento y la propia

arquitectura de la red en los resultados obtenidos.

El siguiente capítulo 5 amplía el modelo a más dimensiones considerando variaciones en las propiedades másicas del UAV y analiza la bondad de su comportamiento. Por otro lado, explora maneras más sencillas de obtener buenos resultados (comparándolas con las definidas en el capítulo 4) y acaba presentando los que se obtienen al tratar de reproducir una situación real (con un menor número de controladores de referencia).

Y, por último, el capítulo 6 recoge las principales conclusiones extraídas del estudio, discute posibles mejoras a tener en cuenta y propone líneas de investigación adicionales para trabajos futuros.

Capítulo 2

Modelo de UAV

2.1. Introducción

En el análisis del comportamiento de nuestra aeronave se van a emplear técnicas de simulación. En este capítulo se desarrolla el modelo de avión que se utilizará en capítulos posteriores para aplicar las distintas técnicas de control.

Se comienza con una visión general de lo que compone un modelo dinámico de un avión según la Mecánica del Vuelo (sección 2.2). En la sección 2.3 se particulariza este modelo para el caso longitudinal de estudio y se presentan las fuerzas y momentos implicados. Además, la última parte de esta sección incluirá modificaciones al modelo desarrollado para casos derivados de una posición variable del centro de masas. La sección 2.4 recoge los parámetros concretos del UAV que se va a estudiar y la 2.5 muestra la implementación del modelo en una función en Matlab. Por último, en la sección 2.6 se determina la envolvente de vuelo del UAV.

2.2. Modelo dinámico general

Para el estudio del control de una aeronave se parte de un modelo dinámico del comportamiento del sistema. En esta sección se presentan las consideraciones generales y los principales pasos para llegar a un modelo de la dinámica longitudinal de un avión. Para un estudio detallado y que considere en profundidad todos los aspectos aquí omitidos se recomiendan Gómez Tierno et al. (2012) para más información sobre la deducción de las ecuaciones y el detalle de los sistemas de referencia, y Stevens et al. (2015) para la descripción de las derivadas aerodinámicas y de estabilidad.

Como es común en los textos de Mecánica del Vuelo, se emplean los **sistemas de referencia** *ejes horizonte local*, *ejes cuerpo* y *ejes viento*, todos ellos centrados en el centro de masas del avión. Los ejes horizonte local, F_h, son unos ejes paralelos a los *ejes tierra* que existirían en el punto subavión (en la superficie terrestre). Los ejes cuerpo del avión, F_b, se mueven ligados a él como sólido rígido. Y el sistema de ejes viento, F_w, se

define en función de la orientación del vector velocidad aerodinámica (la velocidad relativa entre la aeronave y el aire en que se mueve).

La **actitud del avión** se describe en función de las relaciones angulares entre los ejes cuerpo y los ejes horizonte local. Estos tres ángulos son: ψ (ángulo de guiñada), θ (ángulo de asiento) y ϕ (ángulo de balance). Las fuerzas y momentos aerodinámicos suelen ir relacionados con las dos rotaciones existentes entre los ejes cuerpo y los ejes viento: α (ángulo de ataque) y β (ángulo de resbalamiento). Por último, en la descripción de la trayectoria del centro de masas del sólido entran las relaciones angulares entre los ejes viento y los horizonte local: χ (ángulo de guiñada de la velocidad), γ (ángulo de asiento de la velocidad) y μ (ángulo de balance de la velocidad).

Para llegar al modelo dinámico del movimiento del avión se parte de los dos teoremas fundamentales de la Mecánica Clásica. En primer lugar se encuentra el **teorema de la cantidad de movimiento**:

$$\boldsymbol{F} = \frac{\mathrm{d}(m\boldsymbol{V})}{\mathrm{d}t} \tag{2.1}$$

donde $\boldsymbol{F}$ son las fuerzas exteriores, m es la masa del avión, y $\boldsymbol{V}$ es la velocidad absoluta del centro de masas del avión (en ejes inerciales).

Por otro lado está el **teorema del momento cinético**:

$$\boldsymbol{M} = \frac{\mathrm{d}(I\boldsymbol{\omega})}{\mathrm{d}t} \tag{2.2}$$

donde $\boldsymbol{M}$ es la resultante de momentos exteriores alrededor del centro de masas, I es el tensor de inercia y $\boldsymbol{\omega}$ es la velocidad angular absoluta del avión; por lo que $I\boldsymbol{\omega} = \boldsymbol{h}$ es el momento cinético total del avión.

Como es costumbre para el estudio dinámico de un avión, se decide proyectar dichos teoremas en el sistema de ejes cuerpo. Para ello, recordando que estos ejes no son inerciales (pues se moverán respecto a lo que podrían considerarse unos ejes *lo suficientemente inerciales* centrados en la superficie de la Tierra) y mediante el Teorema de Coriolis, los teoremas de cantidad de movimiento y del momento cinético adquieren, respectivamente, las siguientes formas:

$$\boldsymbol{F} = m(\frac{\partial \boldsymbol{V}}{\partial t} + \boldsymbol{\omega} \times \boldsymbol{V}) \tag{2.3}$$

$$\boldsymbol{M} = \frac{\partial (I\boldsymbol{\omega})}{\partial t} + \boldsymbol{\omega} \times (I\boldsymbol{\omega}) \tag{2.4}$$

Sean las componentes en ejes cuerpo de la fuerza total, el momento total, la velocidad lineal absoluta y la velocidad angular absoluta las siguientes:

$$\boldsymbol{F} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}; \quad \boldsymbol{M} = \begin{bmatrix} L \\ M \\ N \end{bmatrix}; \quad \boldsymbol{V} = \begin{bmatrix} u \\ v \\ w \end{bmatrix}; \quad \boldsymbol{\omega} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}. \tag{2.5}$$

Operando en las expresiones (2.3) y (2.4) según (2.5) se llega a las **Ecuaciones de**

Euler del movimiento del avión:

$$\begin{aligned} X &= m(\dot{u} - rv + qw) \\ Y &= m(\dot{v} + ru - pw) \\ Z &= m(\dot{w} - qu + pv) \\ L &= I_x\dot{p} - J_{xz}\dot{r} + (I_z - I_y)qr - J_{xz}pq \\ M &= I_y\dot{q} + (I_z - I_x)pr + J_{xz}(p^2 - r^2) \\ N &= I_z\dot{r} - J_{xz}\dot{p} + (I_x - I_y)pq + J_{xz}qr \end{aligned} \tag{2.6}$$

donde se ha tenido en cuenta que el tensor de inercia es constante respecto a los ejes cuerpo y que el avión es simétrico respecto al plano $x_b - z_b$[1].

Se puede, además, expresar las fuerzas y momentos en función de los términos gravitatorios, aerodinámicos y propulsivos como:

$$\begin{aligned} \boldsymbol{F} &= \boldsymbol{F}_G + \boldsymbol{F}_A + \boldsymbol{F}_T \\ \boldsymbol{M} &= \boldsymbol{M}_A + \boldsymbol{M}_T \end{aligned} \tag{2.7}$$

donde la fuerza debida a la gravedad se proyecta con facilidad en el sistema de referencia horizonte local ($\boldsymbol{F}_G = mg\ \hat{\boldsymbol{k}}_h$). Así, haciendo uso de la matriz de rotación entre estos ejes y los ejes cuerpo y descomponiendo fuerzas y momentos en sus distintas contribuciones según (2.7), el sistema (2.6) resulta finalmente en el **sistema de ecuaciones dinámicas del movimiento completo del avión**:

$$\begin{aligned} -mg\sin\theta + X_T + X_A &= m(\dot{u} - rv + qw) \\ mg\cos\theta\sin\phi + Y_T + Y_A &= m(\dot{v} + ru - pw) \\ mg\cos\theta\cos\phi + Z_T + Z_A &= m(\dot{w} - qu + pv) \\ L_T + L_A &= I_x\dot{p} - J_{xz}\dot{r} + (I_z - I_y)qr - J_{xz}pq \\ M_T + M_A &= I_y\dot{q} + (I_z - I_x)pr + J_{xz}(p^2 - r^2) \\ N_T + N_A &= I_z\dot{r} - J_{xz}\dot{p} + (I_x - I_y)pq + J_{xz}qr \end{aligned} \tag{2.8}$$

2.3. Modelo dinámico longitudinal

Conocido el modelo general, como se adelantaba en la introducción del presente capítulo, se analizará únicamente la dinámica longitudinal del sistema. Con ello, todos los términos lateral-direccionales son idénticamente nulos en la condición de *vuelo simétrico* que se va a estudiar. Por tanto, todas las variables transversales del sistema (2.8) de balance y de guiñada (velocidades lineales y angulares, fuerzas y momentos) se anulan en lo que sigue de este estudio.

[1]Por otro lado, también se ha denotado la derivada de una variable con respecto al tiempo con un punto sobre la propia variable: $\dot{x} = \mathrm{d}x/\mathrm{d}t$.

2.3.1. Sistema de ecuaciones

Particularizando las expresiones de los términos aerodinámicos y propulsivos para el caso longitudinal, es sencillo llegar a:

$$\begin{aligned} -mg\sin\theta + T + L\sin\alpha - D\cos\alpha &= m(\dot{u} + qw) \\ mg\cos\theta - L\cos\alpha + D\sin\alpha &= m(\dot{w} - qu) \\ M_A &= I_y\dot{q} \end{aligned} \tag{2.9}$$

donde T es el empuje, L la sustentación, D la resistencia aerodinámica y M es el momento aerodinámico alrededor del eje y cuerpo (el *momento de cabeceo*). También vuelve a parecer el ya presentado ángulo de ataque, α. Aquí se ha tenido en cuenta que la resultante del empuje de la aeronave se encuentra alineada con el eje longitudinal del avión (eje x cuerpo) y que pasa por su centro de masas (y, por tanto, no da momento). Además de estas ecuaciones dinámicas, se tiene también la relación cinemática entre el ángulo de asiento y la velocidad angular de cabeceo:

$$\dot{\theta} = q \tag{2.10}$$

Por otro lado, las componentes en ejes cuerpo de la velocidad del sistema (2.9) se relacionan con el módulo de la velocidad aerodinámica según:

$$\begin{aligned} u &= V\cos\alpha \\ w &= V\sin\alpha \end{aligned} \tag{2.11}$$

y, consecuentemente, el ángulo de ataque resulta:

$$\alpha = \arctan\left(\frac{w}{u}\right) \tag{2.12}$$

2.3.2. Fuerzas y momentos aerodinámicos

Las fuerzas y momentos aerodinámicos introducidos en (2.9) se escriben, como marca la costumbre, en función de los coeficientes de sustentación, C_L, resistencia, C_D, y momento de cabeceo, C_m:

$$\begin{aligned} L &= \frac{1}{2}\rho V^2 S C_L \\ D &= \frac{1}{2}\rho V^2 S C_D \\ M &= \frac{1}{2}\rho V^2 S \bar{c} C_m \end{aligned} \tag{2.13}$$

donde se introducen la densidad del aire, ρ (como parte de la presión dinámica[2], $q = 1/2\rho V^2$), la superficie alar, S, y la cuerda media aerodinámica, $\bar{c}$. Nótese que se ha eliminado el subíndice «Aerodinámico» (A) de la expresión del momento (ya que $M = M_A$).

Para la descripción del **coeficiente de resistencia** se escoge su polar parabólica

[2] A no confundir con la velocidad angular de cabeceo, también q.

expresada en función del ángulo de ataque:

$$C_D = C_{D0} + C_{D\alpha}\alpha + C_{D\alpha^2}\alpha^2 \tag{2.14}$$

Para el caso de los coeficientes de sustentación y de momento de cabeceo, además de coeficientes estáticos, este modelo *dinámico* debe incluir los correspondientes coeficientes de las *derivadas aerodinámicas* para recoger ciertos términos amortiguadores de las ecuaciones (véase Stevens et al. (2015), página 80). Así, el **coeficiente de sustentación** adquiere la siguiente forma:

$$C_L = C_{L0} + C_{L\alpha}\alpha + \frac{\bar{c}}{2V}C_{Lq}q \tag{2.15}$$

donde, como se comentaba, además de los comunes coeficientes estáticos (C_{L0} y $C_{L\alpha}$), se incluye sustentación proporcional a la velocidad angular de cabeceo (C_{Lq}).

De la misma forma, para el caso del **coeficiente de momento de cabeceo** se tiene:

$$C_m = C_{m0} + C_{m\alpha}\alpha + C_{m\delta e}\delta_e + \frac{\bar{c}}{2V}(C_{mq}q + C_{m\dot{\alpha}}\dot{\alpha}) \tag{2.16}$$

donde dichas derivadas aerodinámicas dinámicas se materializan en los coeficientes C_{mq} y $C_{m\dot{\alpha}}$.

Asimismo, para evaluar estos términos hará falta calcular la derivada del ángulo de ataque en función de las componentes de la velocidad, que, según (2.12), se expresa:

$$\dot{\alpha} = \frac{u\dot{w} - w\dot{u}}{V^2} \tag{2.17}$$

donde se ha tenido en cuenta que la velocidad verifica:

$$V^2 = u^2 + w^2 \tag{2.18}$$

Empleando esta última relación también se llega a que la derivada de la velocidad en función de sus componentes en ejes cuerpo es:

$$\dot{V} = \frac{u\dot{u} + w\dot{w}}{V} \tag{2.19}$$

Por último, el modelo también tiene que recoger variaciones en altura por lo que se ha de añadir la relación:

$$\dot{h} = V \sin\gamma \tag{2.20}$$

donde se recuerda que el ángulo de asiento de la velocidad verifica $\gamma = \theta - \alpha$ para el vuelo simétrico considerado.

2.3.3. Fuerza de empuje

La última fuerza del sistema (2.9) que queda por caracterizar, el empuje de la hélice que propulsa al UAV, se expresa en función del *coeficiente de tracción*, definido como:

$$C_T = \frac{T}{\rho n^2 d^4} \tag{2.21}$$

que relaciona la altura (mediante la densidad, ρ), las revoluciones del motor, n, y el diámetro de la hélice, d, con el parámetro de avance de la hélice, J, según:

$$C_T = C_{T0} + C_{TJ} J \tag{2.22}$$

que, a su vez, se expresa en función de la velocidad de vuelo como:

$$J = \frac{V}{nd} \tag{2.23}$$

Con esto último y todo lo anterior se ha obtenido un modelo longitudinal completo a falta de especificar los valores numéricos de las magnitudes y parámetros implicados.

2.3.4. Modificaciones por centro de gravedad variable

Como últimos añadidos al modelo, se quiere considerar la influencia de permitir la variación de la posición longitudinal del centro de masas y la modificación del momento de inercia longitudinal. Se debe tener en cuenta que, dado el nivel de complejidad del estudio, se van a considerar estos dos aspectos como causas independientes, es decir, que una variación de la posición del centro de masas es independiente del valor del momento de inercia, y viceversa. Resulta evidente que desplazar el centro de masas hacia el morro del avión (o hacia la cola), también influiría en el valor del momento de inercia, al estar dejando *más masa* delante o detrás del eje y cuerpo del avión. Sin embargo, dado que no se conocen con tanto nivel de detalle las propiedades másicas y geométricas del UAV de estudio, se permitirán variaciones de una magnitud que no influyan en la otra.

Un momento de inercia variable, por tanto, solo modifica el valor que toma I_y en la tercera ecuación de momentos del sistema (2.9). Para la variación de la posición del centro de masas sí ha de introducirse alguna consideración adicional.

En primer lugar, como bien es sabido, el índice de estabilidad estática longitudinal frente a perturbaciones en ángulo de ataque, $C_{m\alpha}$, depende inherentemente de la posición del centro de masas. Sin embargo, y como se descubrirá en la siguiente sección, en vez de definir dicho coeficiente a partir de las pendientes de las curvas de sustentación de ala y cola para expresarlo en función del punto neutro del avión (véase Gómez Tierno et al. (2012), capítulo 10), se proporcionará un valor numérico de $C_{m\alpha}$ para una posición del centro de masas de referencia y se añadirán las modificaciones que se muestran a continuación.

En primer lugar, un desplazamiento longitudinal del centro de masas respecto a la

posición de referencia introduciría un momento adicional no contemplado por el coeficiente de referencia producido por las fuerzas aerodinámicas. Concretamente, si se mide el brazo de acción de estas fuerzas como la separación longitudinal entre el centro de masas instantáneo y el de referencia ($\Delta x_{cg} = x_{cg} - x^*_{cg}$), las componente de las fuerzas que daría momento sería la fuerza normal, de coeficiente C_Z dado por:

$$C_Z = -C_D \sin\alpha - C_L \cos\alpha \tag{2.24}$$

que es la proyección de las fuerzas aerodinámicas en la dirección z cuerpo (positiva hacia abajo). Por tanto, este momento adicional se incluye a la expresión del coeficiente de momentos (2.16) con un nuevo término de la forma:

$$C_m = C_{m0} + C_{m\alpha}\alpha + C_{m\delta e}\delta_e + \frac{\bar{c}}{2V}(C_{mq}q + C_{m\dot{\alpha}}\dot{\alpha}) + C_Z\frac{\Delta x_{cg}}{\bar{c}} \tag{2.25}$$

Por otro lado, la potencia de control longitudinal, $C_{m\delta e}$, que aparece en esta definición del coeficiente de momentos también sufre modificaciones. Este término depende de la posición del centro de masas a través del coeficiente de volumen del estabilizador horizontal:

$$V_t = \frac{S_t l_t}{S\bar{c}} \tag{2.26}$$

donde $l_t = x_{cpt} - x_{cg}$ es la distancia entre el centro de presiones de la cola horizontal y la posición del centro de masas.

El análisis de esta variación del centro de masas, Δx_{cg}, resulta en que la nueva potencia de control del avión queda:

$$C'_{m\delta e} = C_{m\delta e}\frac{l^*_t - \Delta x_{cg}}{l^*_t} \tag{2.27}$$

donde l^*_t es dicha distancia entre el centro de presiones del estabilizador horizontal y el centro de masas en la posición de referencia ($l^*_t = x_{cpt} - x^*_{cg}$).

Por último, se considerará también la posibilidad de que la propia masa del UAV cambie: que, de nuevo, solo se traduce en un cambio en su valor numérico en las ecuaciones.

2.4. Parámetros del modelo de UAV

Definido el modelo a emplear, en esta sección se recogen los valores numéricos de todas las magnitudes físicas y coeficientes que definen el problema y que intervendrán en las posteriores simulaciones del comportamiento del UAV de estudio.

Se van a utilizar valores basados (aunque ligeramente modificados) al ya presentado modelo de avión *Skywalker*. Una referencia visual de este avión se encontraba en la figura 1.1. Así, las siguientes tablas recogen todos los parámetros geométricos y másicos (Tabla 2.1), los relativos al motor (Tabla 2.2) y los aerodinámicos (Tabla 2.3).

Por otro lado, se considera una aceleración gravitatoria constante, $g = 9,81 m/s^2$, y la

Tabla 2.1: Parámetros geométricos y másicos.

Parámetro	Valor
Superficie alar, S [m^2]	0.4121
Cuerda media aerodinámica, $\bar{c}$ [m]	0.2192
Masa de referencia, m [kg]	2.5
Posición del centro de masas de referencia[3], $\hat{x}^*_{cg}$ [–]	0.33
Brazo de la potencia de control de referencia, l^*_t [m]	0.80
Momento de inercia de referencia, I_y [kg m^2]	0.156794

Tabla 2.2: Parámetros y coeficientes del grupo motopropulsor.

Parámetro	Valor
Diámetro de la hélice, d [m]	0.254
Revoluciones máximas del motor, REV_{max} [rev/s]	222
Coeficiente de tracción, C_{T0} [–]	0.13805
Coeficiente de tracción, C_{TJ} [–]	-0.2049

Tabla 2.3: Parámetros y coeficientes aerodinámicos.

Parámetro	Valor
Coeficiente de sustentación	
C_{L0} [–]	0.405
$C_{L\alpha}$ [rad^{-1}]	5.379
C_{Lq} [rad^{-1}]	11.134
Coeficiente de resistencia	
C_{D0} [–]	0.0277
$C_{D\alpha}$ [rad^{-1}]	0.14897
$C_{D\alpha^2}$ [rad^{-2}]	0.9848
Coeficiente de resistencia	
C_{m0} [–]	0.0418
$C_{m\alpha}$ [rad^{-1}]	–6.0
$C_{m\delta e}$ [rad^{-1}]	–1.552
$C_{m\dot{\alpha}}$ [rad^{-1}]	–7.0
C_{mq} [rad^{-1}]	–18.4222

variación de la densidad del aire en función de la altura que dicta la Atmósfera Estándar Internacional.

2.5. Modelo en Matlab

Con en modelo definido y los parámetros concretados, se puede implementar el *modelo no lineal de UAV* en Matlab. El código de esta función y más detalle sobre su estructura se encuentra en el listing A.1 del apéndice A.

Dicha función recibe los vectores vector de estado y de control en el instante actual y devuelve el vector de la derivada del vector de estado. A su vez, acepta también otro argumento para incluir la masa del UAV, su posición del centro de masas y su momento

de inercia de cabeceo. Para más detalle sobre estos vectores de estado y de control véase el capítulo 3 sobre las técnicas de control.

2.6. Envolvente de vuelo

Conocidos los parámetros concretos del UAV y definido el modelo se puede obtener en el diagrama altura-velocidad la región en donde el avión puede mantener la condición de vuelo horizontal rectilíneo uniforme y equilibrado: es decir, su *envolvente de vuelo*.

En primer lugar, atendiendo al **equilibrio estático de fuerzas verticales**:

$$L = W \tag{2.28}$$

y eliminando las derivadas temporales de la expresión (2.15) se obtiene que el ángulo de ataque de equilibrio en vuelo horizontal satisface:

$$\alpha = -\frac{C_{L0}}{C_{L\alpha}} + \frac{2W}{\rho V^2 S C_{L\alpha}} \tag{2.29}$$

Observando las fotografías disponibles de un aeromodelo como el de estudio (como la figura 1.1), se puede estimar que los perfiles que conforman el ala pueden asemejarse a un NACA 2410 (véase NASA (2017) y la figura 2.1).

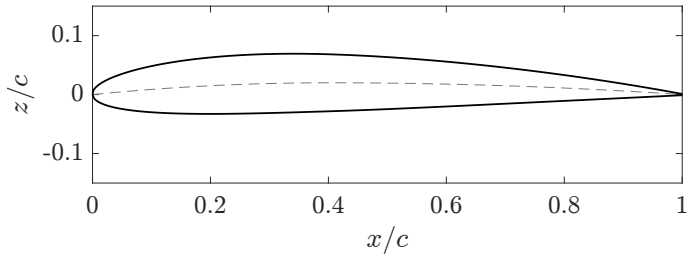


Figura 2.1: Perfil aerodinámico NACA 2410.

Considerando una velocidad de referencia del orden de 10 m/s, la cuerda media aerodinámica (tabla 2.1) como una longitud característica y la viscosidad cinemática del aire a una altitud de referencia de 3000 m (ßengel et al. (2006), Tabla A-9) de $\nu = 1{,}252 \cdot 10^{-5}\,\mathrm{m^2/s}$, se puede estimar que, como mínimo, el vuelo de nuestra aeronave se estará dando a un **Número de Reynolds** en el ala de

$$Re = \frac{V\bar{c}}{\nu} \approx 10^5$$

Así, si se considera la **curva del coeficiente de sustentación frente al ángulo de ataque** de este NACA 2410 para distintos números de Reynolds, se podría establecer un ángulo de entrada en pérdida de referencia, α_{max}. Tomando las curvas de entre $Re = 1 \cdot 10^5$ y $Re = 5 \cdot 10^5$ de un programa como XFOIL (figura 2.2) se puede concluir que un valor

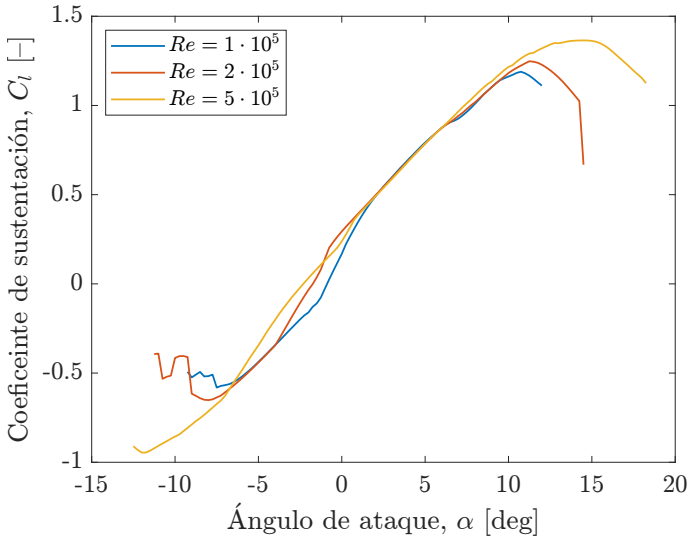


Figura 2.2: Curva del coeficiente de sustentación del NACA 2410 en función del ángulo de ataque para distintos números de Reynolds[4].

razonable para el ángulo de ataque máximo de los perfiles sería:

$$\alpha_{max} = 10^\circ$$

Y, aunque el ángulo de entrada en pérdida del ala sería menor al anterior, se van a considerar iguales por el carácter más que conservador que han tenido los valores utilizados para el cálculo del Reynolds (ya que un Re mayor, como será el caso en la mayoría de los vuelos del UAV, conduce a una entrada en pérdida a mayor ángulo: figura 2.2).

La figura 2.3 representa la ecuación (2.29) en función de la velocidad y para distintas altitudes de vuelo. Se muestra además el valor obtenido para el ángulo de entrada en pérdida. Se puede anticipar pues que, según este criterio, las velocidades mínimas del UAV estarían próximas a los 10 m/s.

Por otro lado, para que este vuelo horizontal fuera posible debe cumplirse el **equilibrio de momentos de cabeceo**:

$$M = 0 \tag{2.30}$$

por lo que, de nuevo, eliminando de la ecuación (2.16) las derivadas temporales, despejando la deflexión del timón de profundidad e introduciendo la relación para el ángulo de ataque, ec. (2.29), se llega a que la deflexión del timón de profundidad cumple:

$$\delta_e = -\frac{C_{m0}}{C_{m\delta e}} - \frac{C_{m\alpha}}{C_{m\delta e}}\alpha = (-\frac{C_{m0}}{C_{m\delta e}} + \frac{C_{m\alpha}}{C_{m\delta e}}\frac{C_{L0}}{C_{L\alpha}}) - \frac{C_{m\alpha}}{C_{m\delta e}C_{L\alpha}}\frac{2W}{\rho V^2 S} \tag{2.31}$$

En este caso, los valores límites para esta deflexión no atienden a criterios aerodinámicos, sino a los meramente geométricos del actuador de la aeronave. Concretamente, el UAV

[4]Datos obtenidos de `airfoiltools.com`:
http://airfoiltools.com/polar/details?polar=xf-naca2410-il-100000,
http://airfoiltools.com/polar/details?polar=xf-naca2410-il-200000,
http://airfoiltools.com/polar/details?polar=xf-naca2410-il-500000

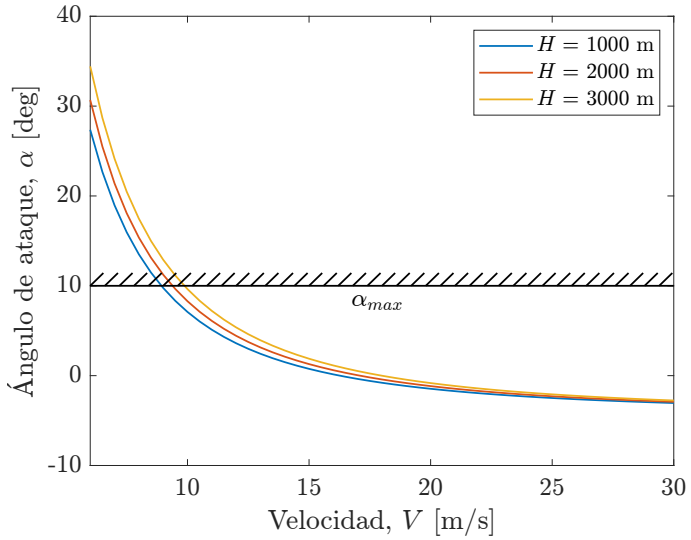


Figura 2.3: Ángulo de ataque para vuelo rectilíneo horizontal en función de la velocidad y para distintas alturas.

puede mover el timón 0.5 rad en ambos sentidos. La figura 2.4 presenta esta deflexión de equilibrio en función de la condición de vuelo y dichos valores de deflexión máxima y mínima.

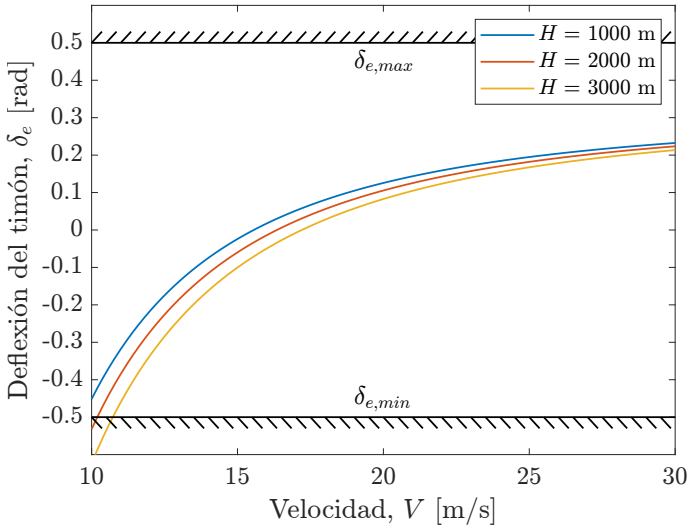


Figura 2.4: Deflexión del timón de profundidad para vuelo rectilíneo horizontal equilibrado en función de la velocidad y para distintas alturas.

Con estos dos análisis se pueden determinar las dos primeras fronteras de la envolvente de vuelo:

- *Entrada en pérdida:* Igualando la expresión para el ángulo de ataque (2.29) a α_{max} se obtiene la velocidad de pérdida en función de la altura.
- *Mínima deflexión de timón de profundidad:* A su vez, igualando la deflexión del timón de equilibrio (2.31) a la deflexión mínima, δ_{min}, se obtiene otra ecuación que relaciona altura y velocidad para este segundo caso límite.

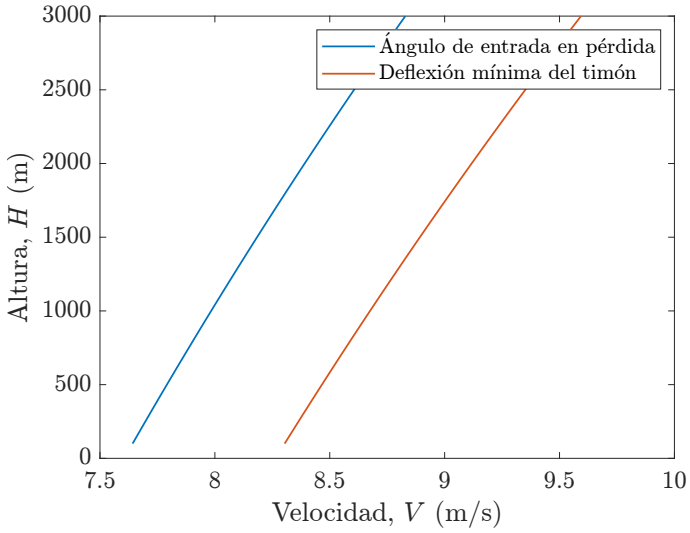


Figura 2.5: Fronteras de la envolvente de vuelo a bajas velocidades.

Con ello, en la figura 2.5 se presentan dichas fronteras. Se puede comprobar que la restricción del caso del timón de profundidad es la predominante, así que será la que dicte la mínima velocidad de vuelo para cada altura. En relación a esta última, se confirma que, efectivamente, la caída de la densidad con la altura penaliza ambos requerimientos.

Con todo lo anterior, se tomará la mínima deflexión del timón como la frontera *izquierda* de la envolvente de vuelo, la correspondiente a las mínimas velocidades.

En segundo lugar, se tiene el **equilibrio de fuerzas horizontales**:

$$T = D \tag{2.32}$$

Atendiendo a las definiciones del apartado 2.3, las fuerzas de empuje y resistencia se escriben:

$$T = \rho n^2 d^4 (C_{T0} + C_{Tj} \frac{V}{nd}) \tag{2.33}$$

$$D = \frac{1}{2} \rho V^2 S (C_{D0} + C_{D\alpha} \alpha + C_{D\alpha^2} \alpha^2) \tag{2.34}$$

Así que, de nuevo, introduciendo el valor del ángulo de ataque para el cálculo de la resistencia, expresión (2.29), se puede representar esta fuerza en función de la velocidad. La figura 2.6 muestra lo anterior y la fuerza de empuje para las revoluciones máximas ($n = REV_{max}$) para varias alturas.

Como se puede comprobar, para cada altura, existe una velocidad por encima de la cual no se puede mantener la condición (2.32), pues la resistencia sería superior a lo máximo que podría entregar el motor. Consecuentemente, la frontera *derecha* de la envuelta de vuelo se obtiene de esta última condición y se representa en la figura 2.7. Nótese la escasa variación de velocidad en el eje de abscisas en comparación con las fronteras *izquierdas* (figura 2.5).

Por último, si se comparan las potencias de la fuerza de resistencia y del empuje, se

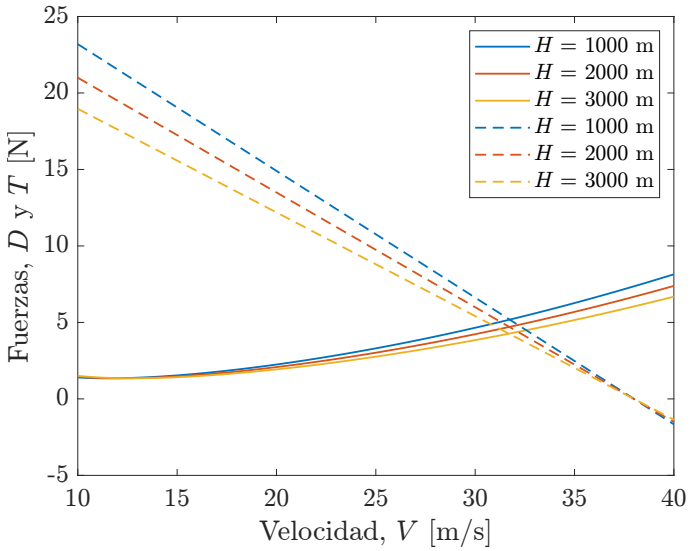


Figura 2.6: Fuerzas de resistencia (D, trazo continuo) y empuje (T, trazo discontinuo) en función de la velocidad y para distintas alturas.

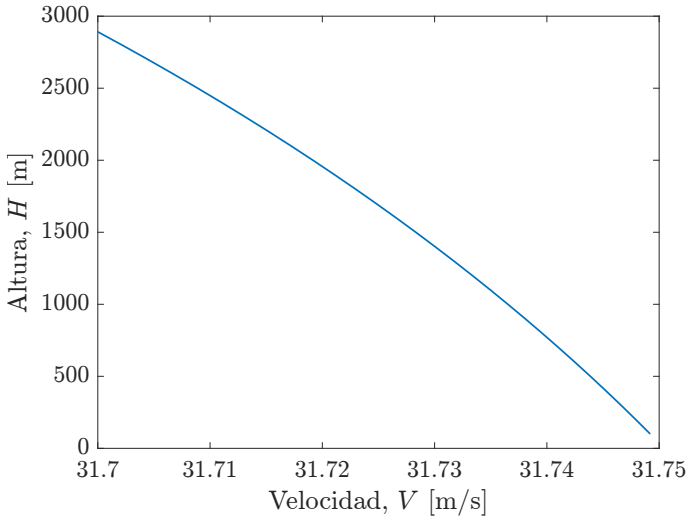


Figura 2.7: Frontera de la envolvente de vuelo a altas velocidades.

comprueba que la potencia disponible del grupo motopropulsor según el modelo empleado excede a la potencia disipada por la resistencia hasta bien superados los 10 km de altitud. Por tanto, para el techo de la aeronave, el límite *superior* de la envolvente de vuelo, se fija, a modo de referencia, en 3000 metros (que ya es una altitud más que considerable tratándose de un aeromodelo de este tamaño). Esta elección es únicamente por intentar imitar lo que sería un aeromodelo de estas características, considerando las capacidades de un sistema de radiocontrol; aunque, como se comentaba, según los valores del modelo, el avión podría subir considerablemente más.

Con todo ello, la figura 2.8 recoge el resultado final para la **envolvente de vuelo de nuestra aeronave**.

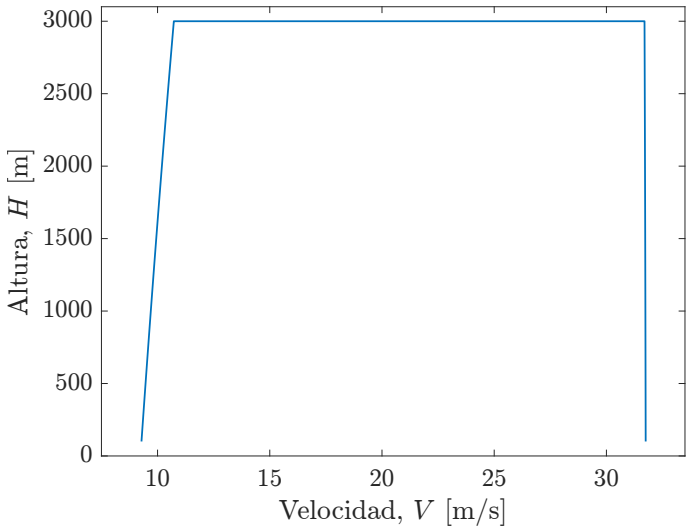


Figura 2.8: Envolvente de vuelo del UAV.

Capítulo 3

Técnicas de control automático

3.1. Introducción

Definido el UAV y su rango de operación, este capítulo da el siguiente paso hacia el objetivo final de este texto de combinar el control de su vuelo con la Inteligencia Artificial. Precisamente, las siguientes secciones sirven para desarrollar las diversas técnicas y métodos de control que desempeñarán el papel de controladores de referencia para comparar y, sobre todo, diseñar y entrenar la red neuronal (siguiente capítulo 4).

Estas técnicas de control se basan en la teoría de control moderno **MIMO (*Multiple Input Multiple Output*)**, en la que la planta del sistema, los bucles y las realimentaciones trabajan, ayudándose del álgebra vectorial, con vectores de estado y de control (en contraposición con el control SISO (*Single Input Single Output*) que, de forma muy resumida, trabajaría con una única variable de entrada y otra de salida). Esta representación vectorial hace realmente sencillo el control del UAV mediante los métodos presentados en las siguientes páginas.

Así, la sección 3.2 concreta la representación multivariable elegida para el sistema (su forma en el *espacio de los estados*). La siguiente, 3.3, presenta el proceso para obtener un modelo lineal alrededor de una posición de equilibrio y, empleando dicho modelo, la sección 3.4 obtiene los autovalores del sistema y su respuesta en bucle abierto bajo distintas condiciones. Concluido el análisis de lo que es el sistema en sí mismo, la 3.5 cierra el lazo de control para diseñar el controlador de referencia por LQR. Y, por último, la sección 3.6 amplía la aplicación del controlador anterior al diseño de varios sistemas de seguimiento.

3.2. Representación en el espacio de los estados

La representación unívoca escogida para describir la situación del sistema en cualquier instante de tiempo, la *representación del UAV en el espacio de los estados* según la teoría MIMO (véase Williams et al. (2007), Capítulo 1), se presenta a continuación. Para la *posición* (recuérdese que se trata de un *vuelo simétrico* en el plano vertical), es suficiente

con especificar la **altura**, H, pues todas las variables implicadas son independientes de un hipotética coordenada horizontal. El cambio de esta posición, la *velocidad*, se representa con el **módulo de la velocidad**, V, y el **ángulo de ataque**, α. Y, por último, la *actitud* de la aeronave queda determinada con el **ángulo de asiento**, θ, y la **velocidad angular de cabeceo**, q. Así, el **vector de estado del sistema** queda definido como:

$$\boldsymbol{x} = \begin{bmatrix} V \\ \alpha \\ \theta \\ q \\ H \end{bmatrix} \tag{3.1}$$

Por otro lado, los mandos disponibles son la **posición del acelerador**, δ_t, (cuyos valores posibles están comprendidos entre 0 «apagado» y 1 «encendido») y la **deflexión del timón de profundidad**, δ_e (que, como ya se ha comentado, existe entre más y menos 0,5 radianes). Por tanto, el **vector de control** queda definido por:

$$\boldsymbol{u} = \begin{bmatrix} \delta_t \\ \delta_e \end{bmatrix} \tag{3.2}$$

El significado de δ_e es meramente la deflexión geométrica del timón demandada en cada momento. Para el caso del acelerador, el parámetro de control δ_t interviene fijando las revoluciones instantáneas del motor, n, de las ecuaciones (2.21) y (2.23) mediante $n = REV_{max}\delta_t$.

3.3. Trimado y linealización

Para parte de las simulaciones del sistema y el desarrollo de las diversas técnicas de control se emplea un modelo linealizado del UAV alrededor de una posición de equilibrio estacionaria. La selección de este punto de equilibrio, el *trimado*, se hace escogiendo las condiciones de vuelo (de altura y velocidad, para vuelo horizontal rectilíneo y uniforme) y empleando la función `trim`[1] de Matlab como se muestra en el Listing A.2 del apéndice A. Este último también incluye la linealización del sistema alrededor de dicha posición de equilibrio mediante la función `linmod`[2] y el diseño del controlador (sección 3.5).

El funcionamiento concreto de estas funciones se aleja del objetivo de estas páginas. `trim` se basa en la optimización cuadrática secuencial (SQP) y `linmod` acude al desarrollo de Taylor a partir de las ecuaciones no lineales de estado y de salida del sistema. Para un estudio más detallado de estos aspectos se recomienda la documentación de `trim` (The MathWorks Inc. (2006b)) y los artículos asociados junto con la de `linmod` (The MathWorks Inc. (2007)) y el detalle matemático de Williams et al. (2007).

[2]Documentación de `trim` en: https://es.mathworks.com/help/simulink/slref/trim.html
[2]Documentación de `linmod` en: https://es.mathworks.com/help/simulink/slref/linmod.html

Con ello, de momento, se obtienen los valores de equilibrio x_{trim} y u_{trim} y las matrices, A, B, C y D, del sistema lineal presentes en las ecuaciones de estado (3.3) y de salida (3.4):

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u} \tag{3.3}$$

$$\boldsymbol{y} = C\boldsymbol{x} + D\boldsymbol{u} \tag{3.4}$$

donde aquí, abusando de la notación, los vectores $\boldsymbol{x}$ y $\boldsymbol{u}$ son las variaciones incrementales de estado y de control, es decir, el valor real de las variables menos su valor en el estado de trimado (y de linealización del sistema).

Así, a modo de ejemplo, si se ejecuta el código del listing A.2 (que trima el sistema a 15 m/s y 1000 m de altura) se obtienen unas matrices del sistema como las siguientes:

$$A = \begin{bmatrix} -0{,}2455 & 6{,}1927 & -9{,}8000 & 0{,}0000 & -0{,}0000 \\ -0{,}0869 & -7{,}4336 & 0{,}0000 & 0{,}8882 & 0{,}0001 \\ 0 & 0 & 0 & 1{,}0000 & 0 \\ 0{,}3202 & -404{,}8883 & -0{,}0000 & -12{,}9709 & -0{,}0002 \\ 0 & -15{,}0000 & 15{,}0000 & 0 & 0 \end{bmatrix} \tag{3.5}$$

$$B = \begin{bmatrix} 7{,}4603 & 0 \\ -0{,}0065 & 0 \\ 0 & 0 \\ 0{,}0239 & -111{,}8166 \\ 0 & 0 \end{bmatrix} \tag{3.6}$$

Por otra parte, la matriz de salida C es la matriz identidad $[I]_{5x5}$ y D es la matriz nula de dimensiones 5x2.

3.4. Respuesta en bucle abierto

Si se toman los **autovalores del sistema** a partir de la matriz A (3.5), se encuentran los siguientes:

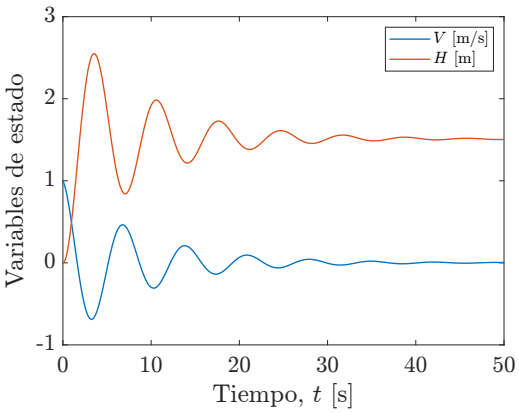
$$\lambda = \begin{Bmatrix} -10{,}2111 + 18{,}7581i \\ -10{,}2111 - 18{,}7581i \\ -0{,}1137 + 0{,}8916i \\ -0{,}1137 - 0{,}8916i \\ -0{,}0003 + 0{,}0000i \end{Bmatrix} \tag{3.7}$$

Los dos primeros, con decaimiento exponencial rápido (parte real) y elevada frecuencia (parte imaginaria), corresponden a los del modo longitudinal **corto periodo**. La siguiente pareja, mucho menos amortiguada y de oscilación más lenta, identifica al modo **fugoide**. Y el último autovalor, no oscilatorio y muy poco amortiguado en comparación con los anteriores, está relacionado con la variación de la altura del UAV. Observando el valor de este

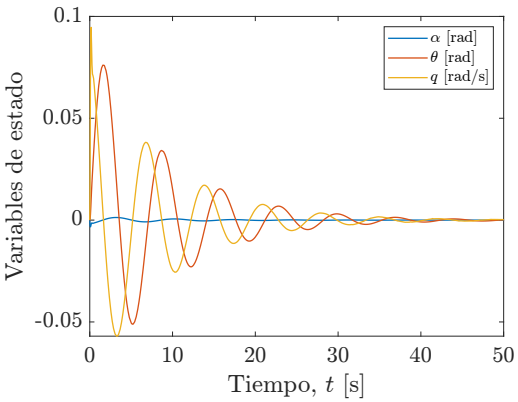
último y antes de simular nada, se puede anticipar un amortiguamiento extremadamente lento en esta variable en comparación con las demás.

Con todo ello, conocidas dichas matrices del sistema, se puede obtener su respuesta temporal frente a una perturbación inicial cualquiera. Así, la figura 3.1 muestra dicha respuesta para un UAV linealizado alrededor de una velocidad de 15 m/s a 1000 metros de altitud, cuando se somete a una perturbación unitaria en velocidad:

$$\boldsymbol{x_0} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$



(a) Velocidad y altura.



(b) Ángulo de ataque, ángulo de asiento y velocidad de cabeceo.

Figura 3.1: Respuesta del UAV en bucle abierto a una perturbación de velocidad de 1 m/s.

Observando las figuras se pueden apreciar con facilidad unas curvas dominadas por el modo fugoide: oscilaciones amortiguadas en velocidad y una escasa variación del ángulo de ataque de perturbación (que caracteriza a este modo propio). Si se observa la evolución de la altura, pudiera parecer que el UAV encuentra una posición de equilibrio a una altura superior a la de trimado (figura 3.1a). Sin embargo, con una simulación que considere más tiempo, figura 3.2 (nótese la escala logarítmica), se puede comprobar que, efectivamente, el sistema vuelve al equilibrio esperado (la situación de trimado). Esta respuesta de la altura casi independiente del resto de variables se debe a que la única relación que existe entre ellas es la debida a la densidad del aire; y, como la variación de esta en alturas del orden del metro es prácticamente nula, la vuelta al equilibrio se da para tiempos del orden de las horas.

Además, por mostrar otra cara del sistema, se estudia la respuesta en bucle abierto a un escalón en el timón de velocidad. Con ello, en la figura 3.3 se muestra la respuesta dinámica del sistema linealizado alrededor de la condición anterior (15 m/s y 1000 m) frente a un escalón en el timón de profundidad de $\delta_e = 0{,}10\,\text{rad} \approx 5{,}7°$. Además, se recoge la respuesta de tres sistemas distintos, en los que se ha variado la posición del centro de gravedad y, por tanto, la potencia de control longitudinal (sección 2.3.4).

En la respuesta a este escalón se observa un salto hasta la nueva velocidad de equilibrio.

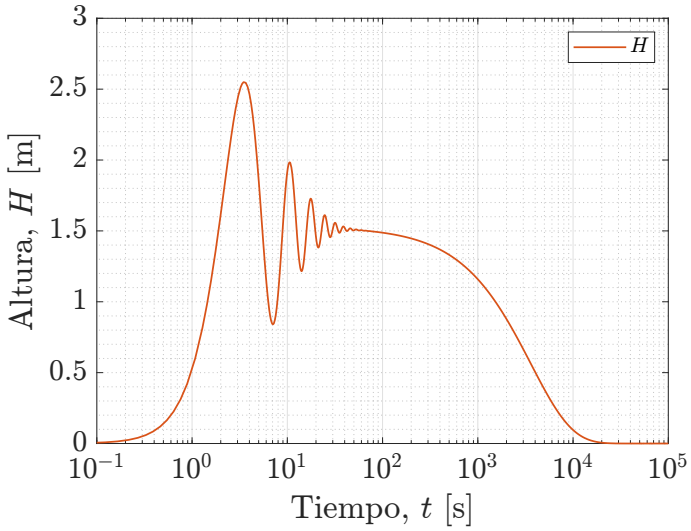


Figura 3.2: Respuesta de la altura del UAV en bucle abierto a una perturbación de velocidad de 1 m/s. Detalle para tiempos mayores.

Esto, debido a que el UAV estaba equilibrado (en el sentido de un coeficiente de momentos nulo) en la posición de trimado, este nuevo incremento positivo de timón lo saca del equilibrio con un momento de picado. Y, por tanto, la velocidad a la que se equilibra es mayor, pues lo que está haciendo el UAV es volar con un ángulo de asiento de la velocidad negativo: esto es, está descendiendo a mayor velocidad que la que mantenía en el vuelo horizontal.

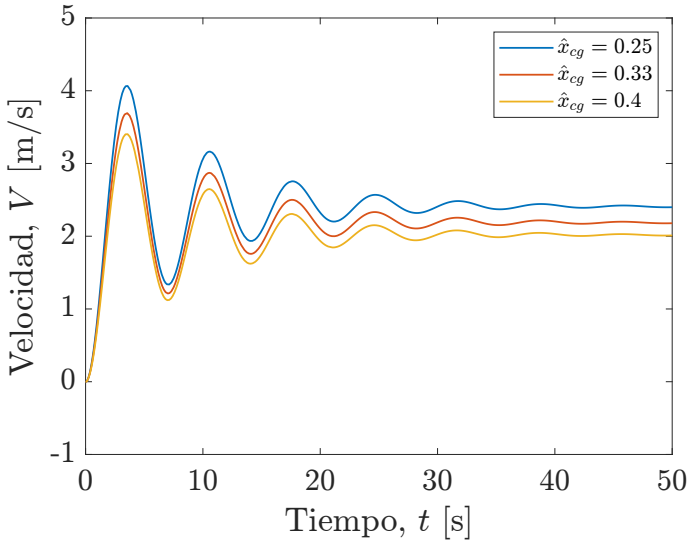


Figura 3.3: Respuesta de la velocidad del UAV en bucle abierto a un escalón en el timón de profundidad de 10 rad.

3.5. Diseño del controlador

Obtenidas las matrices del modelo lineal, se va a emplear el método de control en el dominio del tiempo conocido como **Linear Quadratic Regulator (LQR)** para el

diseño del controlador del sistema. Este método (véase Stevens et al. (2015), Capítulo 5.7), consiste en definir las matrices de costes de estado, Q, y de control, R, para calcular el controlador óptimo, K, que minimiza el error definido como:

$$J = \frac{1}{2}\int_0^\infty (\boldsymbol{x}^T Q\boldsymbol{x} + \boldsymbol{u}^T R\boldsymbol{u})\mathrm{d}t \tag{3.8}$$

donde se recuerda que los vectores $\boldsymbol{x}$ y $\boldsymbol{u}$ denotan incrementos de estas variables respecto a su estado de equilibrio de trimado.

Un ejemplo de cómo se ha implementado esto se encuentra en el ya presentado listing A.2 del apéndice A. Para ello, se emplea la función `lqr`[3] del paquete *Control System Toolbox* de Matlab. De nuevo, para un mayor detalle a cerca de este método se recomienda la anterior referencia Stevens et al. (2015) o la propia documentación de la función The MathWorks Inc. (2006a).

3.5.1. Elección de las matrices de costes del controlador

Las matrices de coste de estado y coste de control se han de escoger tratando de conseguir un controlador que proporcione una respuesta dinámica adecuada.

Se podría escoger inicialmente unas matrices de costes *unitarias* (es decir, $Q = [I]_{5x5}$ y $R = [I]_{2x2}$) para diseñar el controlador por LQR. Sin embargo, debido a los distintos órdenes de magnitud de las variaciones de las componentes del vector de estado, el controlador dedica demasiado esfuerzo en controlar, básicamente, la velocidad y la altura. Esto es debido a que variaciones importantes desde el punto de vista físico en las variables angulares (que pudieran suponer, por ejemplo, pasar cerca de la pérdida por un ángulo de ataque excesivo), al estar multiplicadas por la unidad dentro del coste a minimizar (expresión 3.8), el controlador *no las considera* como variaciones significativas a controlar.

Por ello, es común considerar para la selección de las distintas componentes de la matriz de costes de estado la inversa del cuadrado de la máxima desviación que se quiere permitir en la variable en cuestión. (Esto quiere aprovechar la forma cuadrática que tiene la función de coste, para que en el error final, todas las variables hayan contribuido de una forma similar.)

De este modo, se puede considerar que una variación aceptable para la velocidad sería del orden del metro por segundo y que, para las variables angulares, podría ser una décima de radián. Por tanto, realizando una estimación como la anterior se obtiene una **matriz de costes de estado**:

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 & 0 \\ 0 & 0 & 100 & 0 & 0 \\ 0 & 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 0 & 10 \end{bmatrix} \tag{3.9}$$

[3]Documentación de `lqr` en: https://es.mathworks.com/help/control/ref/lti.lqr.html

Aquí, además, se ha considerado que el coste asociado a la velocidad angular de cabeceo es del mismo orden que el de las variables de ángulo de ataque y de asiento. Y, aunque esto no es exactamente así (pues se encuentra que es común que la velocidad de cabeceo sí alcance valores mayores tras una perturbación), fijar un valor de 100 para el coste de esta variable ayuda a controlar mejor el resto de variables, debido a su relación directa con el ángulo de asiento.

El coste asociado a la variación de la altura, por su parte, se ha dejado en un valor de 10 porque, pese a que las variaciones permitidas pudieran ser incluso de mayor orden que las de la velocidad (y, por tanto, su peso asociado menor), se encuentra que un valor algo más elevado favorece al comportamiento del bucle externo de seguimiento que se desarrolla en el siguiente apartado.

Por otro lado, para la matriz de costes de control, R, suele ser suficiente con fijar un coeficiente que multiplique a la matriz unidad de las dimensiones apropiadas. Considerando la naturaleza angular la deflexión del timón, su rango de variación (entre más y menos 0,5 radianes) y recordando que el acelerador es de orden unidad, este coeficiente podría ser igual a 100. Y, efectivamente, si se fija tal valor de R para un *primer controlador* que use también la matriz de costes de estado ya obtenida, la respuesta de las variables del sistema puede considerarse aceptable. Sin embargo, se encuentra que para proporcionar esta buena respuesta, el timón de profundidad tiene que dar saltos demasiado bruscos (con muy alto valor absoluto de su derivada temporal, es decir, mucha velocidad angular). Por tanto, se decide dejar el coeficiente para el acelerador en 100 y se incrementa el correspondiente a la deflexión del timón, para penalizar su uso. Tras múltiples pruebas con distintos coeficientes y para varias condiciones de perturbación, se llega a una **matriz de costes de control**:

$$R = \begin{bmatrix} 100 & 0 \\ 0 & 500 \end{bmatrix} \tag{3.10}$$

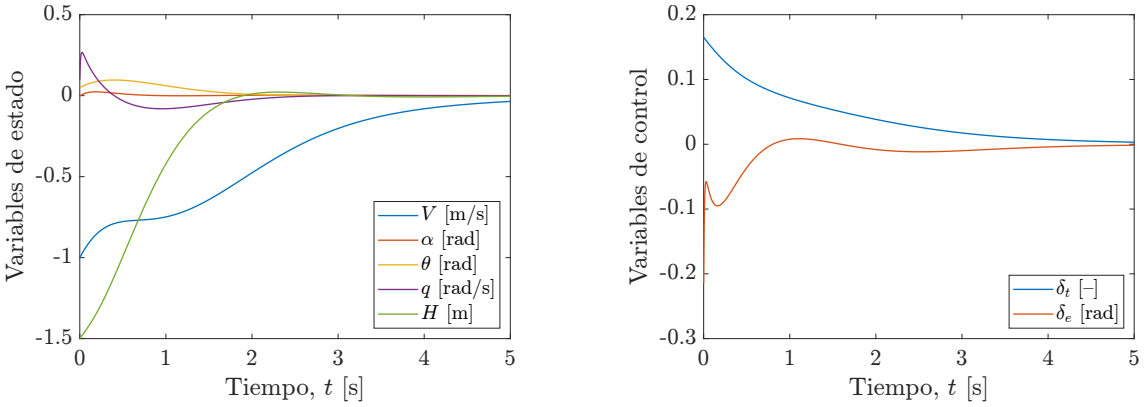
Recordando el listing A.2, si se aplica el método LQR con las matrices Q y R ((3.9) y (3.10)) anteriores al sistema a 15 m/s y 1000 m, las ganancias del controlador óptimo en este punto son:

$$K = \begin{bmatrix} 0{,}1159 & -0{,}5877 & 0{,}8196 & 0{,}0086 & 0{,}0854 \\ -0{,}0229 & 2{,}1773 & -1{,}7712 & -0{,}3428 & -0{,}1361 \end{bmatrix} \tag{3.11}$$

A modo de justificación de lo anterior y como ejemplo del comportamiento del controlador diseñado para el UAV, se presentan las figuras 3.4 y 3.5, cada una correspondiente a un controlador distinto: el que considera mismo coste (100) para ambas variables de control discutido anteriormente («primero») y el controlador definitivo («segundo», 3.11). En ellas, se muestra la respuesta en bucle cerrado del sistema linealizado a 15 m/s y 1000 m a una perturbación (que a partir de ahora será llamada la «perturbación de referencia») como la siguiente:

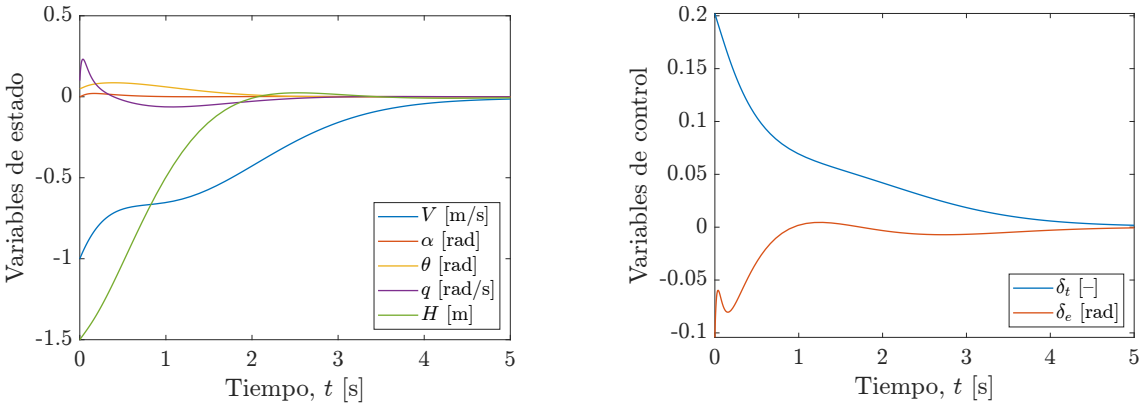
$$\boldsymbol{x_0} = \begin{bmatrix} -1{,}00 & 0{,}00 & 0{,}50 & 0{,}10 & -1{,}50 \end{bmatrix}^T \tag{3.12}$$

Los valores concretos escogidos pretenden ser una perturbación que afecte inicialmente a todas las variables. La diferencia entre las componentes 2 y 3 (α y θ, respectivamente) es para que el UAV tenga una condición inicial de velocidad no horizontal. Y, de la misma forma, el signo negativo para la altura (componente 5) es por someter al sistema a una perturbación medianamente costosa, ya que para una altura incremental positiva, el empuje del motor se anula durante demasiado tiempo ($\delta_t = 0$) y no se obtiene una idea tan clara de cómo de bueno es el controlador.



(a) Estado. **(b)** Control.

Figura 3.4: Respuesta del UAV en bucle cerrado a la perturbación de referencia con el primer diseño del controlador por LQR. Matrices de costes: $Q = \text{diag}(1, 100, 100, 100, 10)$ y $R = \text{diag}(100, 100)$.



(a) Estado. **(b)** Control.

Figura 3.5: Respuesta del UAV en bucle cerrado a la perturbación de referencia con el segundo diseño del controlador por LQR. Matrices de costes: $Q = \text{diag}(1, 100, 100, 100, 10)$ y $R = \text{diag}(100, 500)$.

Comparando las figuras se comprueba lo comentado respecto al timón. La respuesta de las variables de estado para ambos controladores es prácticamente idéntica. Sin embargo, para el primero de los controladores (figura 3.4), el movimiento inicial del timón deflexión recorre 0,15 radianes en dos centésimas de segundo, lo que es un requerimiento excesivo y poco realista para los actuadores de un UAV de estas características (ya que supondría una velocidad angular del timón del orden de 6 rad/s). Si, por el contrario, se emplea el

segundo controlador (figura 3.5), se consigue reducir a una cuarta parte dicha solicitación del timón y que, como se comentaba, apenas se deteriora la respuesta de estado.

3.6. Diseño del sistema de tracking

Para conseguir un piloto automático para la velocidad y altura del UAV, se diseña un *servomecanismo* (véase Williams et al. (2007), capítulo 7.5). Así, se amplía el vector de estado del sistema añadiendo las derivadas de los errores de seguimiento (*tracking*) de las variables a seguir.

En esta sección se revisa ligeramente la matemática detrás del seguimiento de una y dos variables y se termina discutiendo qué valores asignar a las nuevas componentes de las matrices de costes en cada caso para obtener un comportamiento adecuado.

3.6.1. Caso de seguimiento de una única variable

Comenzando para el caso de una variable, por ejemplo la velocidad, la derivada del error de seguimiento es:

$$\dot{\xi} = r - y \tag{3.13}$$

donde y es la variable a seguir, definida a través de la matriz C:

$$y = C\boldsymbol{x} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} V & \alpha & \theta & q & H \end{bmatrix}^T = V \tag{3.14}$$

es decir, y es la velocidad.

El sistema lineal, por tanto, se modifica para quedar de la forma:

$$\begin{Bmatrix} \dot{\boldsymbol{x}} \\ \dot{\xi} \end{Bmatrix} = \begin{bmatrix} [A]_{5x5} & [0]_{5x1} \\ [-C]_{1x5} & [0]_{1x1} \end{bmatrix} \begin{Bmatrix} \boldsymbol{x} \\ \xi \end{Bmatrix} + \begin{bmatrix} [B]_{5x2} \\ [0]_{1x2} \end{bmatrix} \{\boldsymbol{u}\} + \begin{bmatrix} [0]_{5x1} \\ 1 \end{bmatrix} r(t) \tag{3.15}$$

donde A y B son las matrices del sistema original.

Con este nuevo sistema lineal y sus correspondientes matrices ampliadas, se procede con la técnica del LQR para obtener el controlador de la realimentación del bucle cerrado y la constante que multiplica al error en el bucle de tracking. Para ver cómo se conectan entre sí las partes de este nuevo sistema, véase el modelo de Simulink de la figura **??** del apéndice C y el código empleado en el listing A.3 del apéndice A.

Este nuevo estado (el error) debe llevar asociado un coeficiente de coste, q_{ξ_v}, para el método del método LQR, quedando la matriz de costes de estado:

$$Q_t = \text{diag}(1, 1000, 1000, 100, q_{\xi_v}) \tag{3.16}$$

donde el valor de dicho coeficiente se detalla en la sección 3.6.3. La matriz R del método LQR, por su parte, será la misma que la del controlador diseñado en la sección anterior.

Procediendo, como se ha comentado, según el listing A.3, se obtendrían los valores de las K_1 y K_2 necesarias (las que aparecen en el sistema de la figura C.4).

Si, por el contrario, se quisiera diseñar un sistema de seguimiento de la altura (y no de la velociad), se procedería de la misma forma que para el caso de la velocidad, salvo que cambiando la matriz de salida, ya que, en este caso:

$$y = C\boldsymbol{x} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V & \alpha & \theta & q & H \end{bmatrix}^T = H \tag{3.17}$$

es la variable a seguir.

El resto del procedimiento es completamente análogo al del caso para la velocidad y la elección del coeficiente de coste en este caso, q_{ξ_h}, también se recoge en la sección 3.6.3.

3.6.2. Caso de seguimiento de varias variables

Es de interés también poder seguir, por ejemplo, tanto a la velocidad como a la altura. En este caso, se tienen dos errores:

$$\dot{\xi_1} = r_v - V \tag{3.18}$$

$$\dot{\xi_2} = r_h - H \tag{3.19}$$

Por tanto, la ecuación de salida de interés queda:

$$\boldsymbol{y} = C\boldsymbol{x} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} V & \alpha & \theta & q & H \end{bmatrix}^T = \begin{Bmatrix} V \\ H \end{Bmatrix} \tag{3.20}$$

y el sistema ampliado resulta:

$$\begin{Bmatrix} \dot{\boldsymbol{x}} \\ \dot{\boldsymbol{\xi}} \end{Bmatrix} = \begin{bmatrix} [A]_{5x5} & [0]_{5x2} \\ [-C]_{2x5} & [0]_{2x2} \end{bmatrix} \begin{Bmatrix} \boldsymbol{x} \\ \boldsymbol{\xi} \end{Bmatrix} + \begin{bmatrix} [B]_{5x2} \\ [0]_{2x2} \end{bmatrix} \{\boldsymbol{u}\} + \begin{bmatrix} [0]_{5x2} \\ [I]_{2x2} \end{bmatrix} \begin{Bmatrix} r_v(t) \\ r_h(t) \end{Bmatrix} \tag{3.21}$$

donde $\dot{\boldsymbol{\xi}} = \begin{bmatrix} \dot{\xi_1} & \dot{\xi_2} \end{bmatrix}^T$.

Para este caso, la matriz de costes de estado tiene una componente más a añadir (pasando a ser una matriz de 7x7) y siguiendo la notación anterior quedaría:

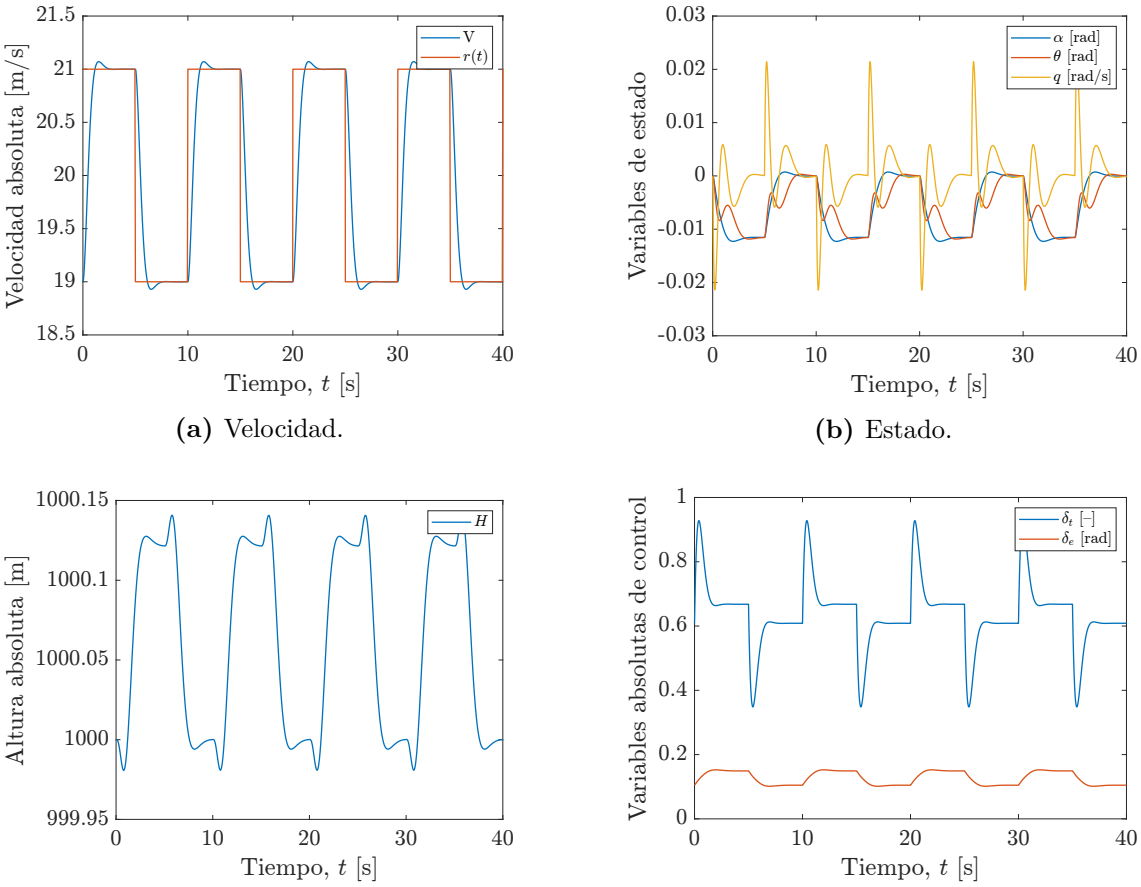
$$Q_t = \text{diag}(1, 1000, 1000, 100, q_{\xi_v}, q_{\xi_h}) \tag{3.22}$$

3.6.3. Obtención de las matrices de costes para cada sistema de tracking

Para un sistema de seguimiento que siga únicamente a la velocidad, vale con fijar el coste asociado en:

$$\xi_{q_v} = 100$$

para obtener resultados más que aceptables. En la figura 3.6 se recoge una representación de lo que hace el sistema (estado y control) cuando se pone a seguir una onda cuadrada para la velocidad de 2 m/s de amplitud y 10 s de periodo alrededor de un trimado a 19 m/s y 1000 m de altitud.



(a) Velocidad. **(b)** Estado.

(c) Altura. **(d)** Control.

Figura 3.6: Respuesta del UAV a un sistema de tracking para la velocidad siguiendo una onda cuadrada. $r(t)$ es la referencia a seguir, figura (a).

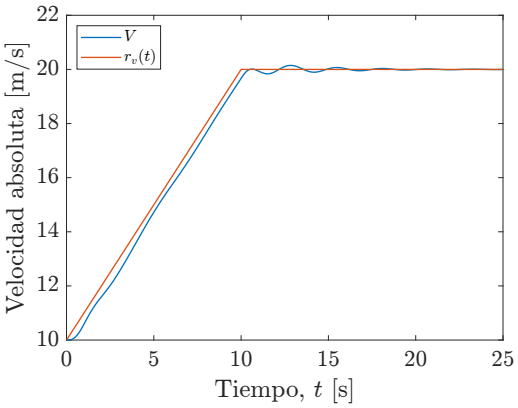
Un sistema de tracking que solo siga a la altura, por otro lado, puede no ser excesivamente útil en estos casos. Esto es porque, por ejemplo, si se pone al sistema a seguir una senda de ascenso con un gradiente determinado, el controlador solo cuida la altura y, como era de esperar, el resto del sistema pierde velocidad y entra en pérdida casi en cuanto se comanda la orden de seguimiento.

Por ello, para seguir la altura, se empleará el caso en el que se haga seguimiento de ambas variables: tanto la altura como la velocidad, para cuidar la actitud global de la aeronave. Para ello, se ha observado que una buena combinación de pesos para el sistema de tracking de las dos variables es:

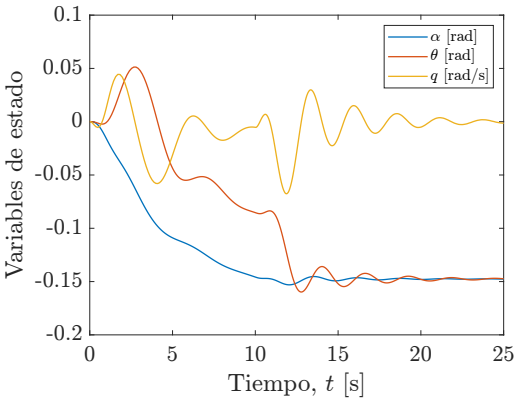
$$\xi_{q_v} = 100$$

$$\xi_{q_h} = 5$$

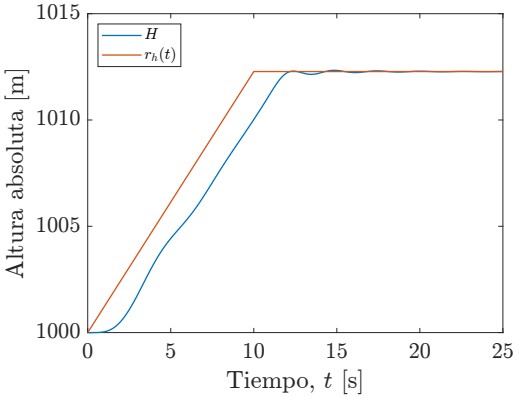
A continuación, en las figuras 3.7 se presenta la respuesta de este sistema de tracking.



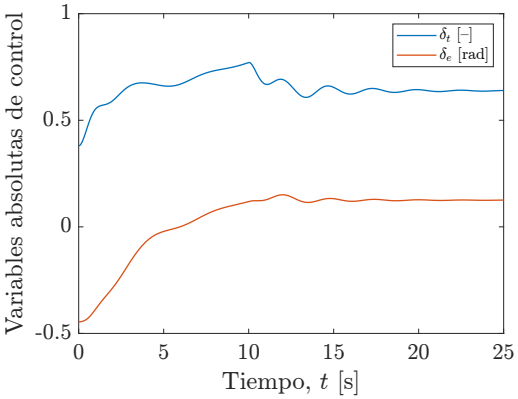
(a) Velocidad.



(b) Estado.



(c) Altura.



(d) Control.

Figura 3.7: Respuesta del UAV a un sistema de tracking de la velocidad y la altura siguiendo una rampa para la velocidad y otra para la altura durante 10 segundos, seguida de un segundo tramo plano. $r_v(t)$ es la referencia a seguir por la velocidad, figura (a), y $r_h(t)$ es la referencia para la altura, figura (c).

En particular, las referencias a seguir son dos rampas de 10 segundos de duración, seguidas de un segundo tramo *plano*. Se considera una rampa en velocidad de 1 metro por segundo y por segundo (una aceleración de 1 m/s^2) que lleva al UAV de los 10 m/s (sitio donde está diseñado el controlador de su bucle interno) hasta los 20 m/s. La rampa para la altura fija un gradiente de subida de 7° desde los 1000 metros hasta algo por encima de los 1010 metros.

Como se puede comprobar, figura 3.7a, el seguimiento de la velocidad resulta ser bastante cercano a la referencia comandada. Para el caso de la altura, figura 3.7c, las diferencias son algo mayores, pero no supone tanto problema en este caso[4]. Para el caso de las variables angulares, figura 3.7c, se puede comprobar que el ángulo de ataque describe, con más o menos oscilaciones, la forma hiperbólica descrita por la teoría (recuérdese la figura 2.3). El ángulo de asiento, por su parte, sigue a esta hipérbola algo desfasada hacia arriba durante los primeros 10 segundos (durante las rampas) para asegurar un ángulo de

[4]En el caso de dar un peso mayor en la matriz de costes de estado al error de seguimiento de la altura se consigue reducir dicho error. Sin embargo, las oscilaciones que este hecho introduce en el ángulo y velocidad de cabeceo no compensa seguir de forma más cercana a la altura.

asiento de la velocidad, $\gamma = \theta - \alpha$, que cumpla el gradiente de subida comandado. Se puede comprobar, por otro lado, las pequeñas oscilaciones generales que induce el «cambio de tramo» a los 10 segundos.

Por último, conviene comentar que esta simulación se ha realizado con el modelo no lineal de UAV, ya que la diferencia de presión dinámica entre los 10 y los 20 m/s hace que estemos lejos de la linealidad que nos ofrecería un modelo lineal trimado a 10 m/s. El controlador empleado, por su parte, estaba diseñado a estos 10 m/s y 1000 m siguiendo el modelo de LQR descrito anteriormente. El capítulo siguiente se dedica a mejorar el comportamiento que pueda ofrecer dicho controlador al variar el punto de operación dentro de la envolvente de vuelo.

Capítulo 4

Diseño del controlador mediante una red neuronal

4.1. Introducción

En el capítulo anterior se han discutido métodos para diseñar controladores y sistemas de seguimiento. Sin embargo, las ganancias de estos controladores están siempre ajustadas al punto de operación que se esté considerando, y es de esperar que un controlador «óptimo» en un punto (el que se obtiene por LQR) no proporcione una respuesta adecuada en otra condición de vuelo. Esto, como ya se adelantaba al final del capítulo anterior, se debe principalmente a la distinta capacidad de control que se tiene a las diferentes velocidades de vuelo (ya que los momentos introducidos por la deflexión del timón son proporcionales a la presión dinámica) y, en mucha menor medida, a la variación de la densidad atmosférica con la altura (y las implicaciones que tiene esto, además de la velocidad, en la capacidad propulsiva de la hélice). Por ello, la evolución del sistema dependería tanto de la condición actual de vuelo, como de la que se haya usado para diseñar el controlador que se esté empleando.

Por tanto, sería de interés encontrar una forma de disponer de un controlador que fuera óptimo (o cercano al óptimo) en cada punto de la envolvente de vuelo. Para ello, se va a diseñar una red neuronal que sirva para inferir las ganancias más adecuadas en cada momento en función de la condición de vuelo (altura y velocidad) a partir de una familia de controladores.

Así, la sección 4.2 se encarga de definir esta noción de «mejor respuesta» y desarrolla una forma de cuantificarlo numéricamente (mediante el Índice de Desempeño). Fijado el criterio para lo que se considera un controlador «óptimo» en cada punto, en la sección 4.3 se detalla el procedimiento para generar el dataset para entrenar a la red y se resumen las principales características implicadas en su diseño. Después, en la sección 4.4 se recogen los resultados del comportamiento de este nuevo controlador frente a una perturbación y en la sección 4.5 se compara su comportamiento en un sistema de seguimiento con otras

soluciones alternativas. Por último, la sección 4.6 recoge consideraciones adicionales sobre las decisiones que se han ido tomando para el diseño de la red neuronal y las implicaciones que tienen las distintas alternativas en todo este proceso.

4.2. Evaluación del Índice de Desempeño

Esta sección se comienza gráficamente. Como se comentaba, un controlador va a funcionar de forma distinta si se saca de su punto de funcionamiento. Con esto, considérese el controlador «A», diseñado para una condición de 15 m/s y 3000 m de altura (punto A), y el controlador «B», para 27 m/s y 500 m (punto B). Se somete al sistema a la perturbación de referencia (3.12). Así, se va a comparar el comportamiento del controlador de ganancias K_A (controlador A) en ambas condiciones de vuelo (puntos A y B) y el comportamiento del sistema con el controlador B (de ganancias K_B) en las mismas dos condiciones.

En primer lugar, la figura 4.1 muestra el comportamiento del sistema en el punto A si se equipa con cada uno de los dos controladores. Se puede observar que la respuesta con el controlador K_B es menos controlada. Como dicho controlador *espera* una mayor presión dinámica, la vuelta al equilibrio de las variables es más lenta y el pico negativo de la velocidad de cabeceo es mucho más exagerada que para el caso del controlador K_A (el «original» en el punto A). En cuanto al control, no se observan diferencias demasiado significativas.

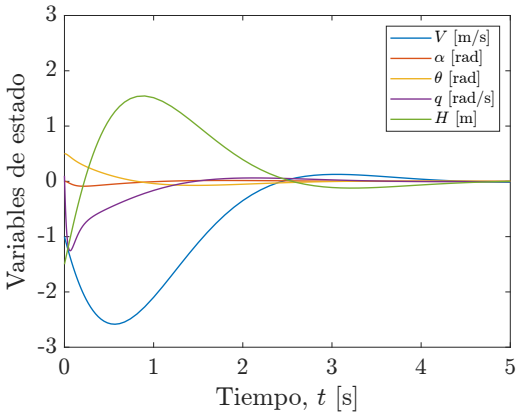
Haciendo lo mismo para el sistema en el punto B se llega a la figura 4.2. En este caso, se puede apreciar que la respuesta que proporciona el controlador K_A presenta unos picos mayores para velocidad y altura que los que se obtienen con el controlador original K_B.

La conclusión de este análisis es que, aunque las diferencias puedan ser sutiles y dependan, evidentemente, de la pareja de puntos de operación que se comparen e incluso de las condiciones iniciales de perturbación, un controlador no va a funcionar exactamente como desearíamos si se aleja demasiado de su punto de diseño (aquí A y B son puntos en extremos opuestos de la envolvente de vuelo, figura 4.4).

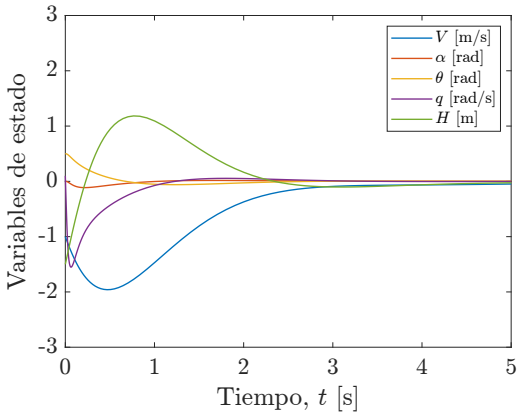
Estas comparaciones, no obstante, se vuelven demasiado tediosas y lentas cuando se quiere hacer un análisis más exhaustivo en toda la envolvente de vuelo. Por tanto, con objeto de tener un método para «evaluar» las respuestas de una manera sistemática y rápida se define el **Índice de desempeño**, ***PI*** (de *Performance Index*). Este PI va a derivar de la teoría de la función de coste del método LQR (expresión 3.8) con unas matrices de costes modificadas definidas como sigue:

$$Q_{PI} = \text{diag}(1, 100, 100, 0, 0) \tag{4.1}$$

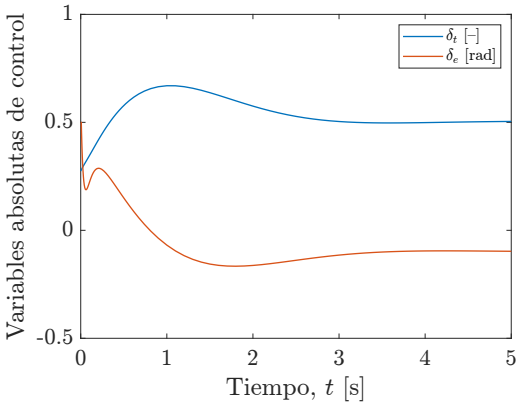
$$R_{PI} = \text{diag}(100, 100) \tag{4.2}$$



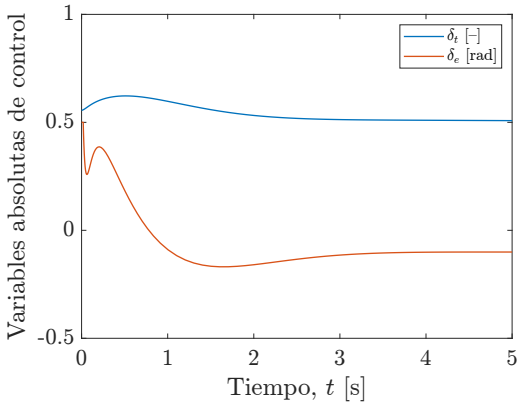
(a) K_A en A, estado.



(b) K_B en A, estado.



(c) K_A en A, control.



(d) K_B en A, control.

Figura 4.1: Respuesta del UAV a una perturbación en el punto de operación A con dos controladores distintos: uno diseñado en el punto A, K_A, y otro en el punto B, K_B. Las figuras (a) y (c) muestran la evolución incremental del estado y absoluta del control, respectivamente, para el sistema con el controlador A, y las figuras (b) y (d) hacen lo propio para el controlador B.

El Índice de Desempeño se define, por tanto, a partir de este error modificado:

$$J_{PI} = \frac{1}{2}\int_0^\infty (\boldsymbol{x}^T Q_{PI} \boldsymbol{x} + \boldsymbol{u}^T R_{PI} \boldsymbol{u}) \mathrm{d}t \tag{4.3}$$

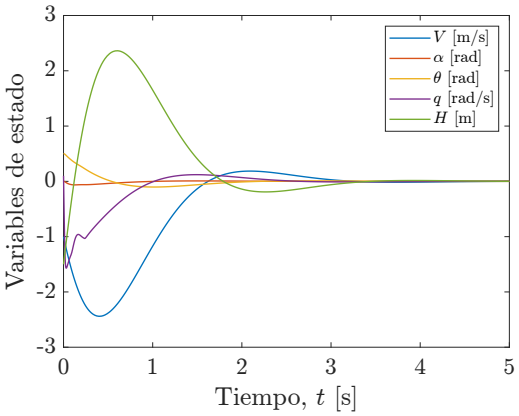
y queda:

$$PI = \frac{1000}{J_{PI}} \tag{4.4}$$

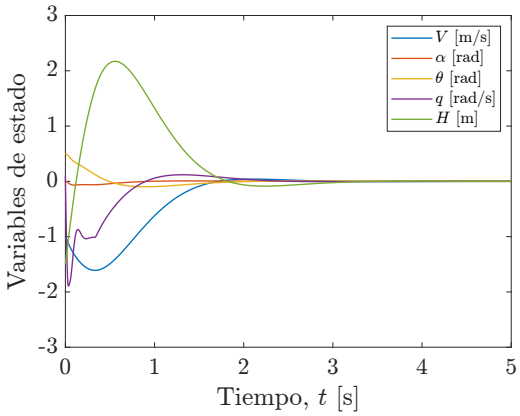
Por tanto, como este PI está definido como inversamente proporcional al error, *un PI mayor indicará una «mejor respuesta»*.

Nótese que, como las simulaciones para evaluar este PI se hacen hasta un tiempo finito, la integral del error deberá evaluarse en este mismo periodo. Esto no supone un problema siempre que se use el mismo tiempo de simulación para todos los casos y este sea suficiente para permitir al sistema volver al equilibrio.

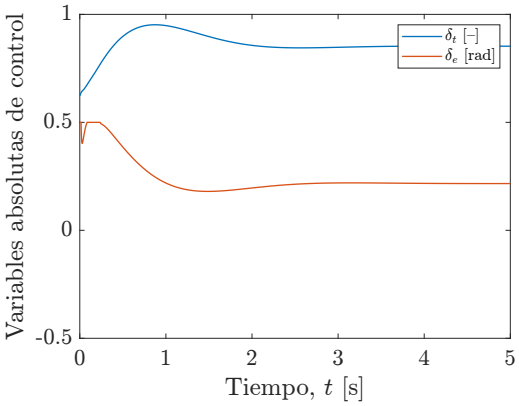
Con ello, la forma de encontrar el PI de un controlador de referencia cualquiera (como podrían ser K_A o K_B de los ejemplos anteriores) es como se muestra en el diagrama de la



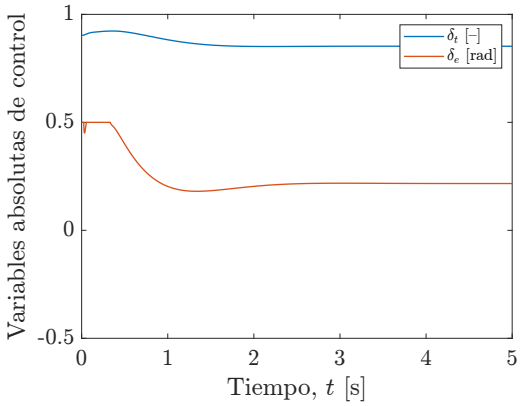
(a) K_A en B, estado.



(b) K_B en B, estado.



(c) K_A en B, control.



(d) K_B en B, control.

Figura 4.2: Respuesta del UAV a una perturbación en el punto de operación B con dos controladores distintos: uno diseñado en el punto A, K_A, y otro en el punto B, K_B. Las figuras (a) y (c) muestran la evolución incremental del estado y absoluta del control, respectivamente, para el sistema con el controlador A, y las figuras (b) y (d) hacen lo propio para el controlador B.

figura 4.3. Se discretiza una malla que recorra la envolvente de vuelo; se escoge un punto de operación (trimado) de dicha malla (V_i, H_j); se obtiene el modelo lineal (matrices A y B); y se realiza una simulación con la perturbación usada en los ejemplos anteriores instalando en el UAV el controlador de referencia que se quiere evaluar (K_{ref}). Conocidas las señales de su respuesta temporal, $x(t)$ y $u(t)$, se evalúa la función de costes modificada (J_{PI}) y se obtiene el PI del controlador en dicho punto.

Siguiendo este procedimiento para los controladores K_A y K_B de esta sección se llega a la figura 4.4. En ella se muestra el Índice de Desempeño de cada uno de los dos controladores en un dominio que contiene a la envolvente de vuelo. También se muestran los puntos de operación A y B.

A partir de las figuras se pueden sacar varias conclusiones acerca de los controladores. En primer lugar, a ambos extremos de las dos figuras, a bajas y altas velocidades, se observan los límites de la envolvente de vuelo: en estas regiones se dan índices mucho menores (es decir, peor comportamiento). Esto es normal, pues se está poniendo al contro-

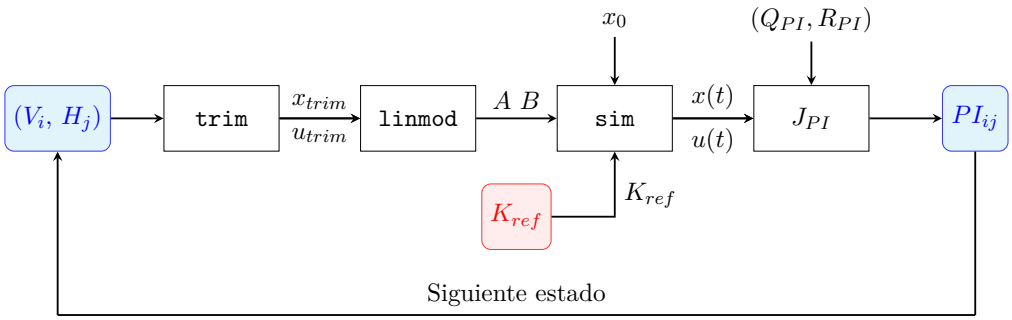
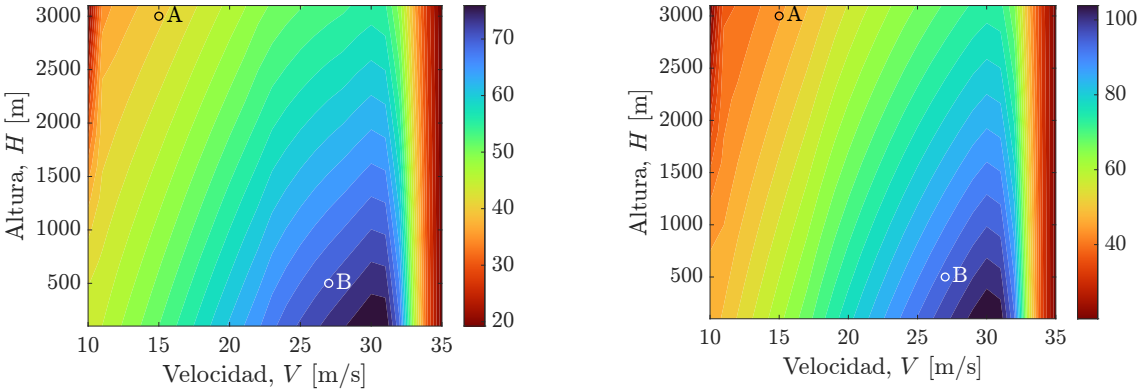


Figura 4.3: Diagrama de flujo para la evaluación del PI en los distintos puntos de la envolvente de vuelo con un controlador de referencia.



(a) PI de K_A.

(b) PI de K_B.

Figura 4.4: Índice de Desempeño frente a la perturbación de referencia (3.12) en toda la envolvente de vuelo con el controlador K_A (a) y con el K_B (b).

lador a intentar corregir una perturbación en una condición de vuelo en la que el UAV no puede siquiera mantener el vuelo horizontal rectilíneo uniforme y equilibrado; ya sea por deflexión del timón (límite izquierdo) o por insuficiencia de empuje (frontera derecha).

Por otro lado, como era de esperar, ambos controladores funcionan mejor a mayores velocidades y bajas alturas, debido a la ya mencionada influencia de la presión dinámica. Además, y siguiendo con lo comentado sobre la figura 4.2, pese a que el controlador A funciona mejor a dichas mayores velocidades, no llega a alcanzar el PI que proporciona en esa zona el controlador B.

Tratando, entonces, de mejorar estas respuestas, se puede diseñar un controlador para cada punto de operación de la envolvente de vuelo. Así, se obtiene una familia de controladores por LQR para cada uno de los puntos de la malla y, con ellos, se puede evaluar su PI. Este nuevo proceso se detalla en la figura 4.5 y el resultado en la 4.6.

Observando este nuevo PI, no obstante, no se encuentran demasiadas diferencias con del controlador B. Este proceso, no obstante, sirve para ilustrar que es posible diseñar un controlador para cada punto de la envolvente de vuelo y es suficiente para considerar una

forma de generar el dataset que se discute en la siguiente sección.

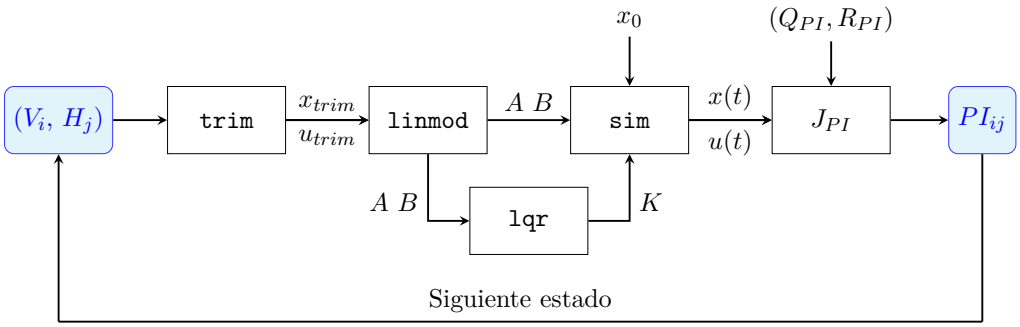


Figura 4.5: Diagrama de flujo para la evaluación del PI en los distintos puntos de la envolvente de vuelo con un controlador diseñado en cada punto.

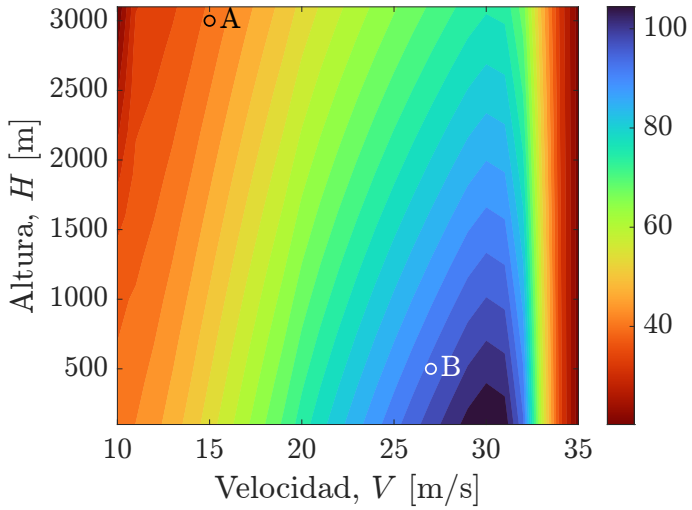


Figura 4.6: Índice de Desempeño frente a la perturbación de referencia (3.12) de un controlador diseñado en cada punto de operación a lo largo de toda la envolvente de vuelo.

Como último comentario, conviene tener en cuenta que este *Performance Index* concreto que se está empleando para evaluar el desempeño de un controlador en cualquier condición de vuelo no es el único posible. Las matrices Q_{PI} y R_{PI} ((4.1) y (4.2)) que se han escogido para obtener el error J_{PI} sirven para tener una idea de cómo se podría evaluar un controlador en un proceso de diseño real. Si para otro problema se tuviera que diseñar un controlador (o una familia de ellos) que diera al UAV un comportamiento determinado, sería tan sencillo como rediseñar este PI para que reflejase dicho comportamiento. Para esto se podría heredar el razonamiento empleado en la elección de las matrices de costes durante el diseño del controlador por LQR (3.5). Si, para una aplicación en particular, el controlador tuviera que reducir al máximo las oscilaciones del ángulo de asiento, un PI que sirviese para evaluar el desempeño del controlador debería tener un peso muy elevado en la tercera componente de la diagonal de Q_{PI} (la asociada al error en θ).

Para otras alternativas de diseño del PI se podría incluir una medida de la derivada

de las señales de estado y de control, pues dan información sobre las aceleraciones del sistema y la solicitación de los actuadores y superficies de control. También se podría prestar atención al tiempo de pico de una determinada variable (para ver cómo de agitada es la respuesta inicial) y al porcentaje de pico, o incluso ampliar el estudio para intentar considerar la cualidades del vuelo de la aeronave. Todo es cuestión de qué criterio se elija para diseñar (y por tanto evaluar) cada controlador.

4.3. Diseño y entrenamiento de la red neuronal

Como ya se ha determinado, el controlador que queremos que el UAV lleve en cada punto es el que nos proporciona el método LQR en cada una de esas condiciones. Se pretende, por tanto, generar una familia de controladores para alimentar a una red neuronal con objeto de que pueda inferir qué ganancias son las más óptimas en cada punto de operación.

Merece especial mención el hecho de que los miembros de esta familia de controladores difieren tan solo sutilmente del resto de miembros. Es decir, las ganancias del controlador en un punto serán ciertamente similares a las de cualquier otro punto de la envolvente. Sin embargo, esta forma de generar los controladores sirve tan solo como ejemplo de las capacidades que tendrá la red de inferir el controlador óptimo para cada instante. En caso de tener otro criterio para diseñar cada controlador en cada punto (siguiendo el razonamiento del final del apartado anterior), el procedimiento de definición de la red es equivalente y solo se modificaría la forma de definir el dataset que se recoge en los siguientes párrafos.

Entrando en la arquitectura de la red neuronal, como se ha dicho, las únicas variables que (de momento) definen cada estado o punto de operación son (1) la velocidad y (2) la altura. Por tanto, las **variables de entrada** de la red deberán ser la pareja de valores velocidad-altura.

El objetivo de la red es, dada la anterior pareja, obtener las ganancias del controlador óptimo para ese estado. Estas ganancias son cada una de las 10 componentes (por las 5 variables de estado y las 2 de control) de la matriz K de realimentación del bucle cerrado. Por tanto, las **variables de salida** de la red son estas 10 ganancias (10 valores numéricos).

4.3.1. Generación del dataset

Y, como no podía ser de otra forma, esta red ha de entrenarse con datos. Y, a diferencia de lo que suele ocurrir en los problemas de Deep Learning, en esta ocasión tenemos la oportunidad de generar nosotros mismos los datos que necesitemos (proceso en la figura 4.7). Se escoge un punto de operación de la malla que discretiza la envolvente de vuelo (V_i, H_j), se realizan los pasos necesarios para llegar al modelo lineal y, con las matrices de

costes ya empeladas con el método LQR, (3.9) y (3.10), se obtiene la matriz de ganancias buscada.

En el apéndice A se clarifica todo este procedimiento con el código de Matlab del listing A.4. Concretamente, la malla que se ha empleado para discretizar la envolvente de vuelo tiene una distancia entre puntos de 0,25 m/s en velocidad y 100 m en altura. A su vez, el rango que se ha considerado es de 10 a 30 m/s en velocidad, y de 100 a 3100 m en altura. Por tanto, este dominio evitar incluir la frontera derecha de la envolvente de vuelo, pues no se obtiene información de valor al analizar controladores en dicha zona.

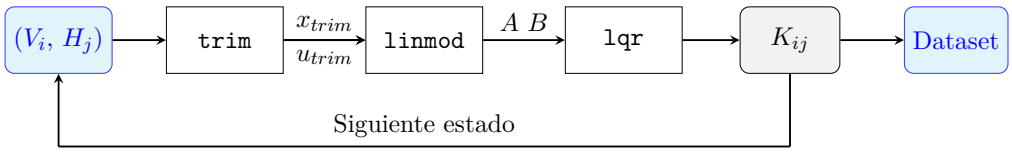


Figura 4.7: Diagrama de flujo para la generación del dataset con las ganancias de los controladores a partir de la condición de vuelo.

4.3.2. Diseño y arquitectura

La inferencia de estas ganancias con una red neuronal a partir de la condición de vuelo no debieran suponer una arquitectura demasiado complicada, pues se trata básicamente de una interpolación entre puntos de operación de la malla más o menos cercanos. Con ello, la estructura de la red se esquematiza en la figura 4.8. Se consideran **4 capas densas ocultas** con unos tamaños sucesivos de 20, 40, 40 y 20 neuronas que conectan las 2 entradas con las 10 salidas. A cada de estas neuronas se asocia una **función de activación «ReLU»** (Rectified Linear Unit), para poder replicar la naturaleza no linear de esta interpolación. La **función de coste** del proceso de optimización será el error cuadrático medio (**Mean Squared Error**) y el algoritmo será un **método Adam**. Se considerará también la desactivación de parte de la red mediante el proceso de **Dropout** con una probabilidad de 0,1 y la posibilidad de hacer «**Early Stopping**» en el proceso de entrenamiento. A modo de referencia, se fija el máximo número de epochs en 500 y la «paciencia» del early stopping en 20.

Para todo este desarrollo de la red neuronal se ha usado principalmente la librería de Deep Learning `PyTorch` de Python y un ejemplo de la implementación de todo lo anterior se puede encontrar en el listing B.1 del anexo B. El significado concreto y las implicaciones de la mayoría de lo discutido en el párrafo anterior se discutirá en la sección 4.6.

Por otro lado, conviene remarcar que las variables de entrada de la red (las parejas de velocidad y altura del dataset) son pasadas por una normalización (según una distribución normal estándar) antes de introducirlas en la red. Esto también se recoge en el listing mencionado y tiene implicación en futuras secciones.

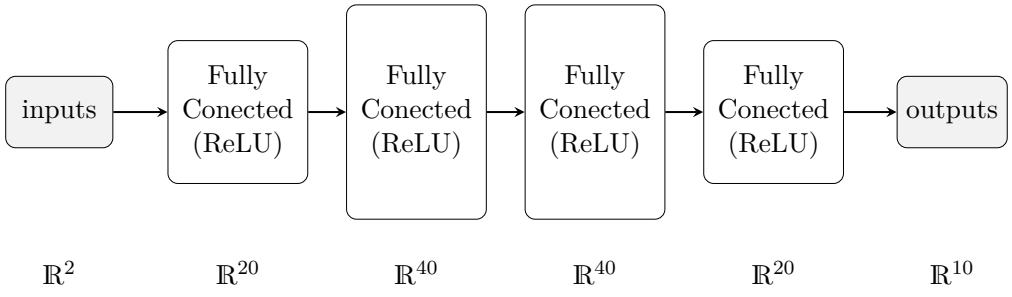


Figura 4.8: Esquema de la arquitectura de la red neuronal.

4.4. Comportamiento del controlador frente a una perturbación

Entrenado el modelo, en primer lugar, se quiere comparar la capacidad de control de las ganancias que devuelve la red dado un punto de operación con la respuesta que proporcionaría un controlador diseñado directamente en dicho punto. Para ello, se escoge un estado que «no haya visto» la red (recuérdese el dataset descrito en la sección 4.3.1), como por ejemplo:

$$\begin{bmatrix} V \\ H \end{bmatrix} = \begin{bmatrix} 20,8 \text{ m/s} \\ 492 \text{ m} \end{bmatrix}$$

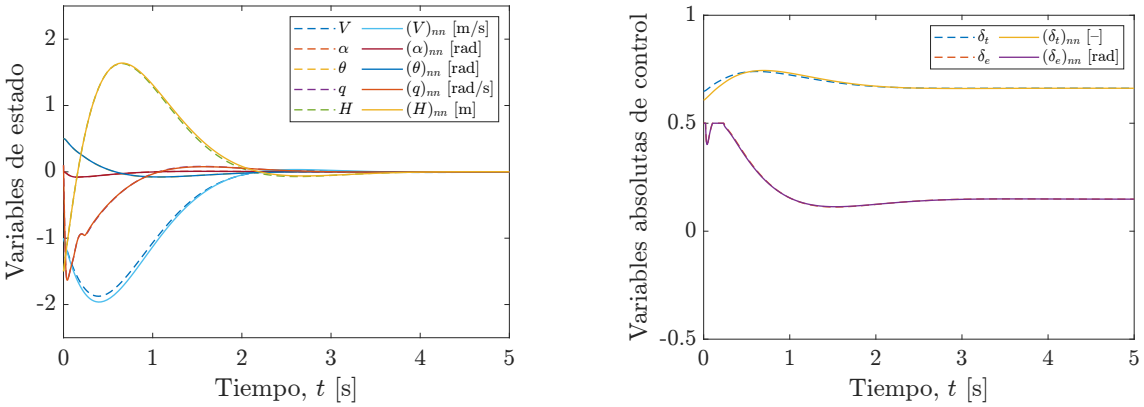
y se somete a la perturbación de referencia (3.12) para comparar sus respuestas.

Así, el trazo discontinuo de la figura 4.9 representa la evolución del sistema que proporcionaría el controlador óptimo diseñado por LQR en este estado y las líneas continuas son la respuesta del controlador inferido por la red en ese punto.

Como se puede comprobar, ambas respuestas son prácticamente idénticas; es decir, la red neuronal ha hecho un buen trabajo obteniendo las ganancias apropiadas para acercarse a las que proporcionaría el controlador «óptimo» en ese punto. Además, para cuantificar cómo de parecidas son se recupera el Índice de Desempeño y se encuentra que en ambos casos también son muy próximos entre sí:

$$(PI)_{lqr} = 75{,}8002; \quad (PI)_{nn} = 74{,}4687$$

Sin embargo, este buen resultado no es tan afortunado en otros puntos de operación. Para ilustrarlo, la figura 4.10 compara el PI del controlador por LQR en cada punto con el que proporciona la red neuronal, para distintas velocidades y siempre a la misma altura (1000 m). Como se puede comprobar, la respuesta de la red se ajusta más a la del controlador óptimo en la zona central del intervalo y hacia los extremos el PI obtenido se aleja del controlador original. No obstante, conviene destacar que, a menores velocidades, la red neuronal consigue superar el PI del controlador original. Lo contrario ocurre para el extremo derecho de intervalo. Es decir, la red consigue suavizar las diferencias de PI



(a) Estado. **(b)** Control.

Figura 4.9: Comparación de la respuesta del estado (a) y control (b) del UAV frente a la perturbación de referencia (3.12) a 20, 8 m/s y 492 m. El trazo discontinuo representa las señales del controlador por LQR en dicho punto y el continuo es la respuesta cuando se emplea la red neuronal.

en la envolvente de vuelo: reduce los PI altos a altas velocidades («deteriorando» el comportamiento), pero aumenta los menores (peores) índices del controlador en vuelos más lentos.

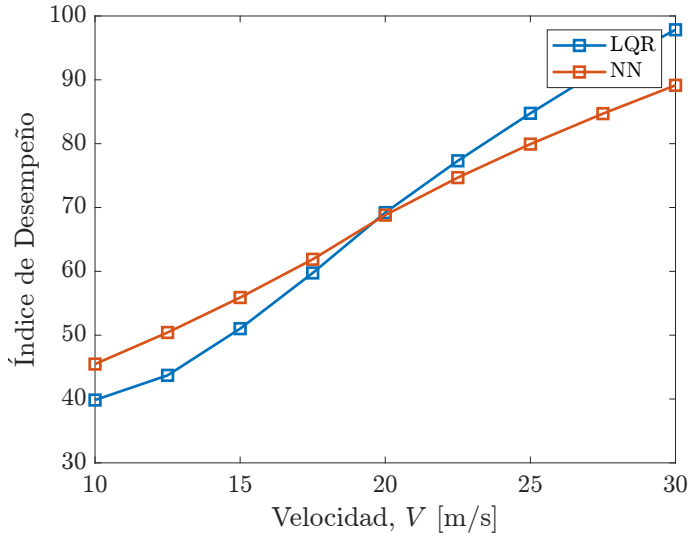
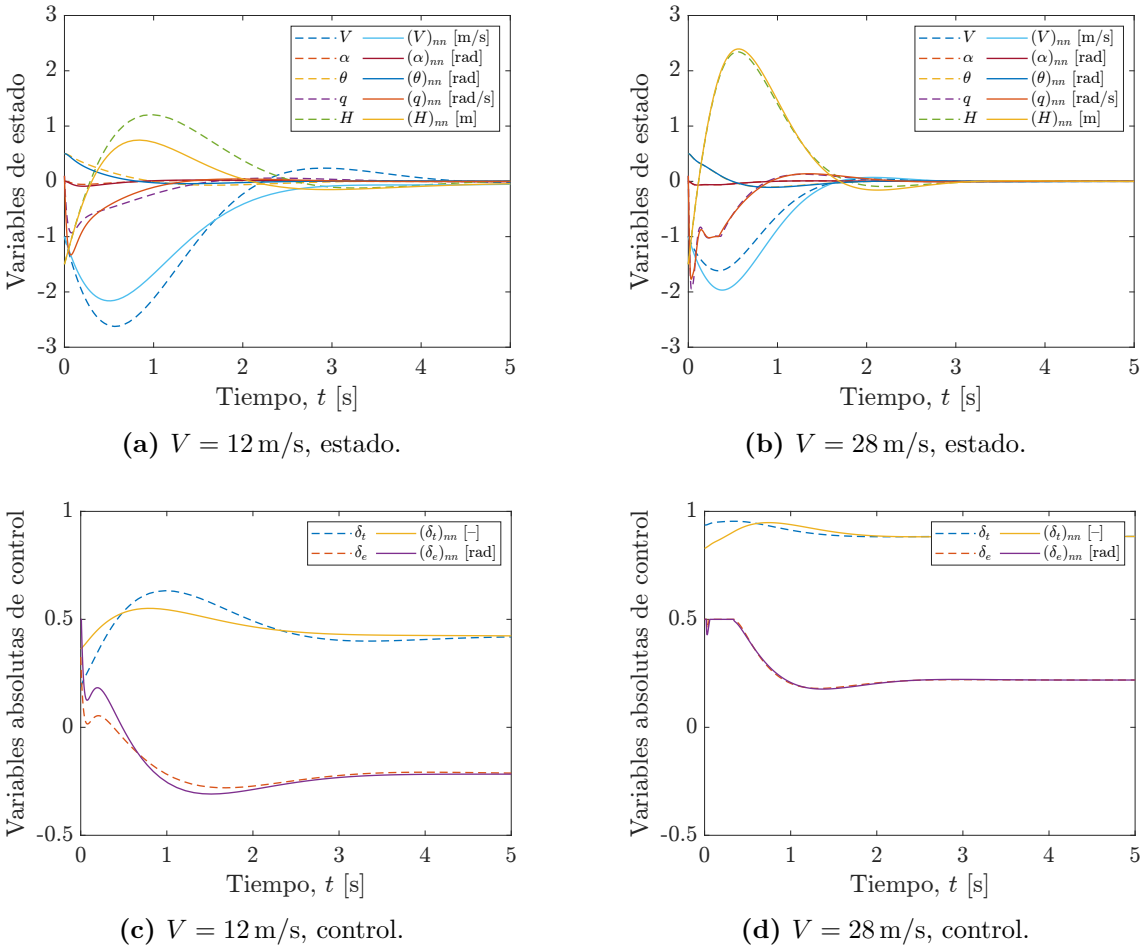


Figura 4.10: Comparación de los Índices de Desempeño frente a la perturbación de referencia (3.12) entre un controlador diseñado en cada punto de operación y el que proporciona la red neuronal, para distintas velocidades y a 1000 metros de altura.

Como ejemplo de la anterior se tiene la figura 4.11, donde se compara las respuestas del controlador original y de la red. Se presentan los casos del UAV a 12 y 28 m/s, ambos a 1000 metros de altura. Se comprueba, como se anticipaba en la figura 4.10, que los picos de las oscilaciones son menores al usar la red en el caso del punto a baja velocidad (12 m/s) y que, sin embargo, la red genera perturbaciones ligeramente mayores cuando opera a 28 m/s.



(a) $V = 12\,\mathrm{m/s}$, estado. **(b)** $V = 28\,\mathrm{m/s}$, estado.

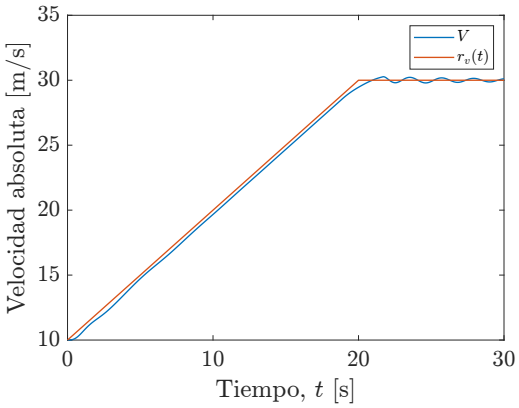
(c) $V = 12\,\mathrm{m/s}$, control. **(d)** $V = 28\,\mathrm{m/s}$, control.

Figura 4.11: Comparación de la respuesta del estado, (a) y (c), y el control, (b) y (d), del UAV frente a la perturbación de referencia (3.12) para 12 m/s (izquierda, (a) y (c)) y 28 m/s (derecha, (b) y (d)). El trazo discontinuo representa las señales del controlador por LQR cada punto y el continuo es la respuesta cuando se emplea la red neuronal.

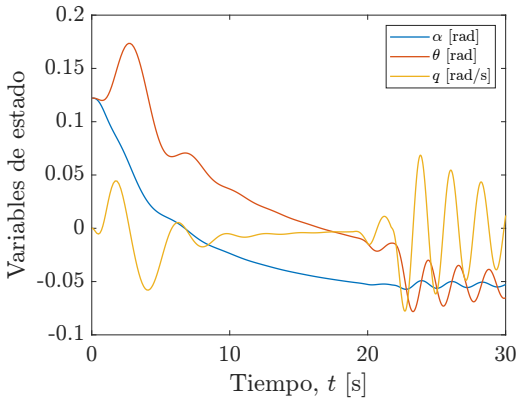
4.5. Comportamiento del controlador en un sistema de tracking

En la sección anterior se muestra cómo de bien consigue replicar al controlador original nuestra red neuronal frente a una perturbación en un vuelo determinado. Sin embargo, este no es el objetivo para el que se ha diseñado la red: si se supiera cuáles van a ser las condiciones exactas de vuelo (es decir, su punto de operación) se usaría el controlador óptimo diseñado para ese punto y no sería necesaria la red en ningún caso. El problema es que una aeronave, y en particular nuestro UAV, debe poder funcionar en una amplia variedad de puntos y, como se anticipaba en la introducción de este capítulo y se demostraba en la sección 4.2, un controlador óptimo en un sitio no tiene porqué serlo en otro.

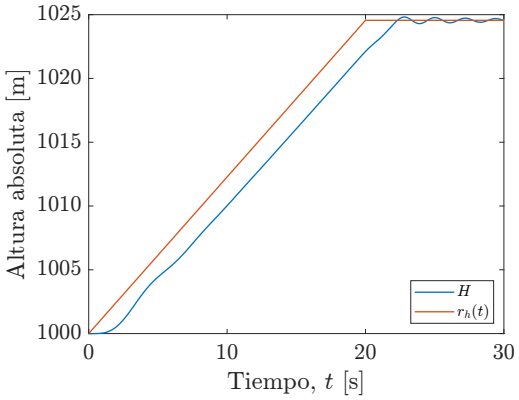
Es por este motivo por lo que se quiere diseñar la red: para poder disponer de una forma de ir cambiando de controlador, dentro de una familia de ellos, a medida que el UAV varía su punto de operación dentro de su envolvente de vuelo.



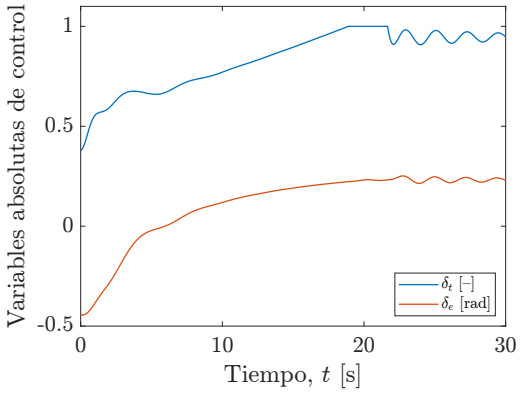
(a) Velocidad.



(b) Estado.



(c) Altura.

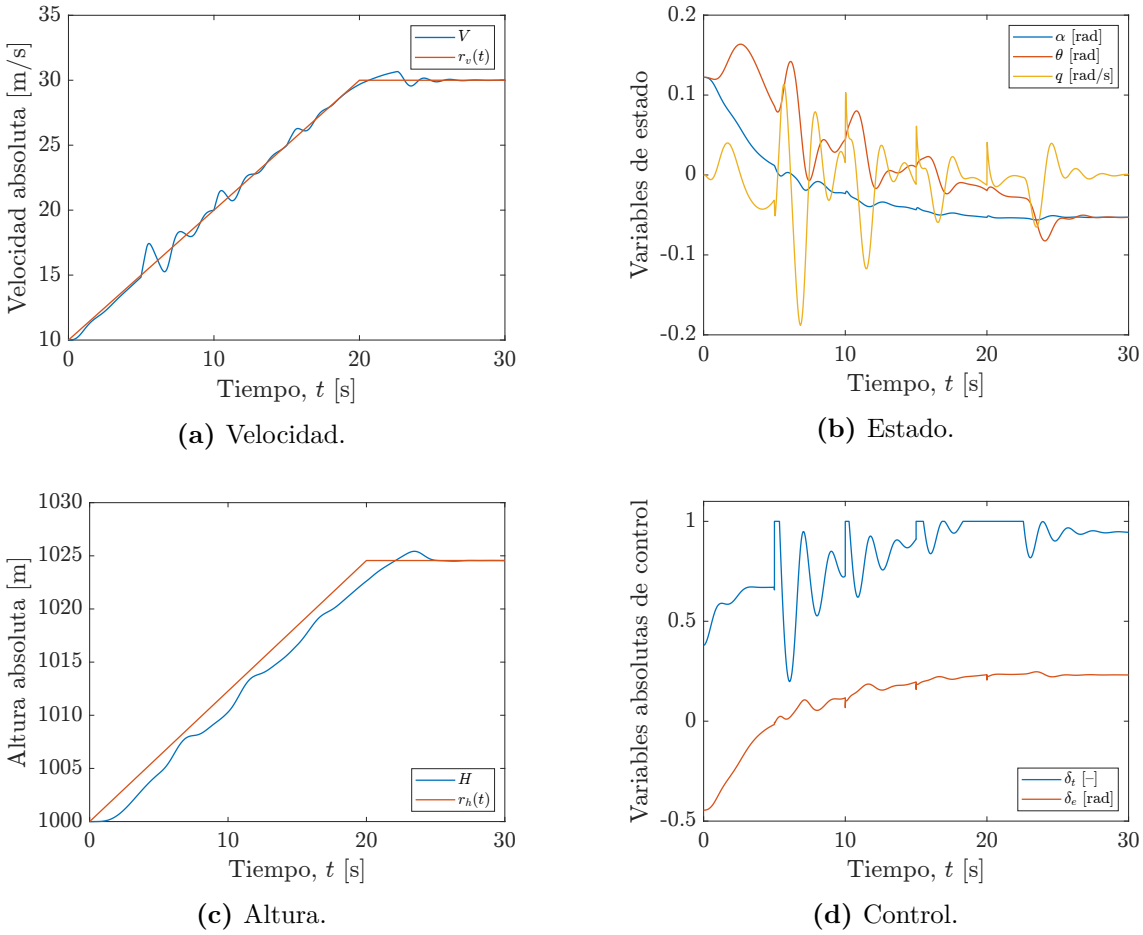


(d) Control.

Figura 4.12: Respuesta del UAV a un sistema de tracking de la velocidad ($r_v(t)$, figura (a)) y la altura ($r_h(t)$, figura (b)) empleando un único controlador diseñado a 10 m/s.

Así, en esta sección se analiza precisamente esto y, para mostrar un ejemplo de variación del punto de operación, se vuelve a recuperar el sistema de seguimiento. Se considera la misma señal de referencia que para la figura 3.7 (dos rampas: 1 m/s^2 en velocidad y 7° de ángulo de ascenso en altura, partiendo de los 10 m/s y los 1000 m). En esta ocasión, no obstante, se mantienen dichas rampas durante 20 segundos (en lugar de los 10 que se emplearon en el diseño del sistema de tracking) antes de llegar a otros 10 segundos de zona plana. En la sección 3.6.3 se mostró el ejemplo de mantener el controlador diseñado en el punto de inicio durante todo el proceso de seguimiento. Si se hace lo mismo en esta ocasión se obtiene la figura 4.12.

Observando los resultados, la primera de las rampas en este caso eleva la velocidad absoluta hasta los 30 m/s, y se puede observar que el controlador diseñado a 10 m/s presenta serios problemas de estabilidad hacia el final de la simulación (visible, por ejemplo, en las amplias oscilaciones de la figura 4.12b). Esto es, entre otras cosas, debido a que el controlador no espera tanta presión dinámica y el movimiento excesivo del timón provoca las oscilaciones que se observan. Claramente, esto es un comportamiento lejos del ideal y, en caso de diseñar una aeronave que fuera a alcanzar tales velocidades, sería necesario obtener alguna forma de reducir dichas oscilaciones.



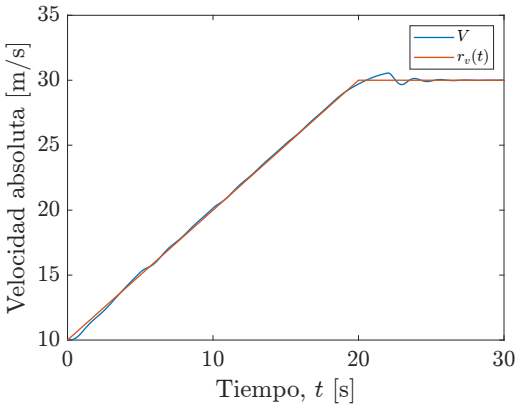
(a) Velocidad. **(b)** Estado.

(c) Altura. **(d)** Control.

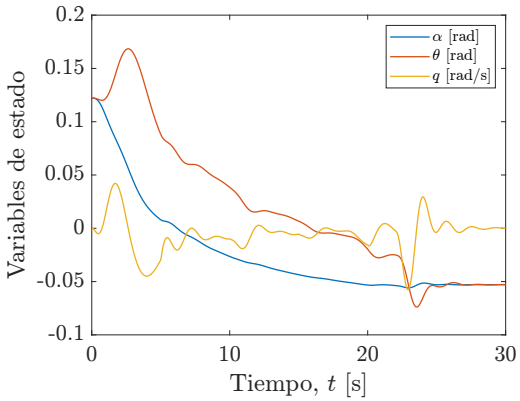
Figura 4.13: Respuesta del UAV a un sistema de tracking de la velocidad ($r_v(t)$, figura (a)) y la altura ($r_h(t)$, figura (b)) empleando una familia de controladores definidos cada 5 m/s.

Una primera forma de «solucionar» este problema de una manera algo burda (y que no se emplea en la realidad) sería definir intervalos de operación (en función de la velocidad y la altura) de una familia de controladores y pasar de un controlador a otro cuando se cambiase de intervalo. Con ello, si se diseña una familia de controladores que dividan el rango de 10 a 30 m/s en 6 intervalos (es decir, un cambio cada 5 m/s) y se trima cada controlador en el centro de su intervalo, ejecutando la misma simulación se llega a la figura 4.13.

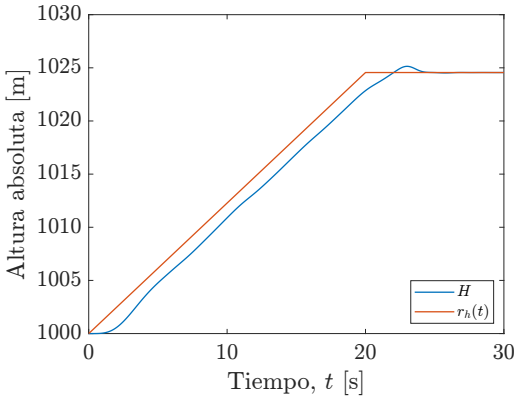
Como se puede observar, no obstante, esta «solución» únicamente empeora el problema, ya que se producen saltos en todas las variables en cada cambio de intervalo, por lo que la respuesta como conjunto es claramente peor y menos controlada. La velocidad sufre pequeñas oscilaciones que deterioran el seguimiento de la velocidad; la altura, no obstante, sigue más o menos una trayectoria similar al caso anterior del controlador único; pero los ángulos y velocidades angulares sí experimentan saltos constantes y el resultado es que durante todo el ascenso y aceleración (las dos rampas) el UAV no ha parado de oscilar y no ha llevado una trayectoria suave y controlada. Es muy revelador el caso de la velocidad de cabeceo: mientras en el caso anterior solo se encontraban oscilaciones significativas en el cambio de pendiente de la referencia ($t = 20\,\text{s}$), en este seguimiento el sistema no de-



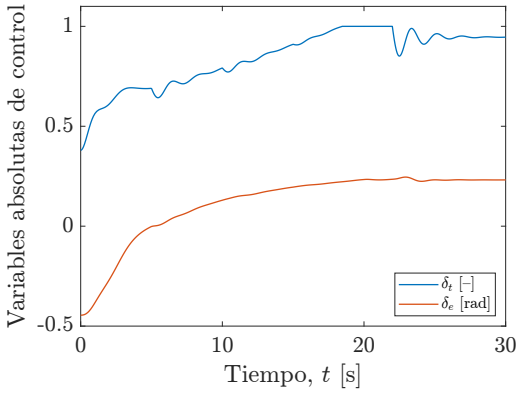
(a) Velocidad.



(b) Estado.



(c) Altura.



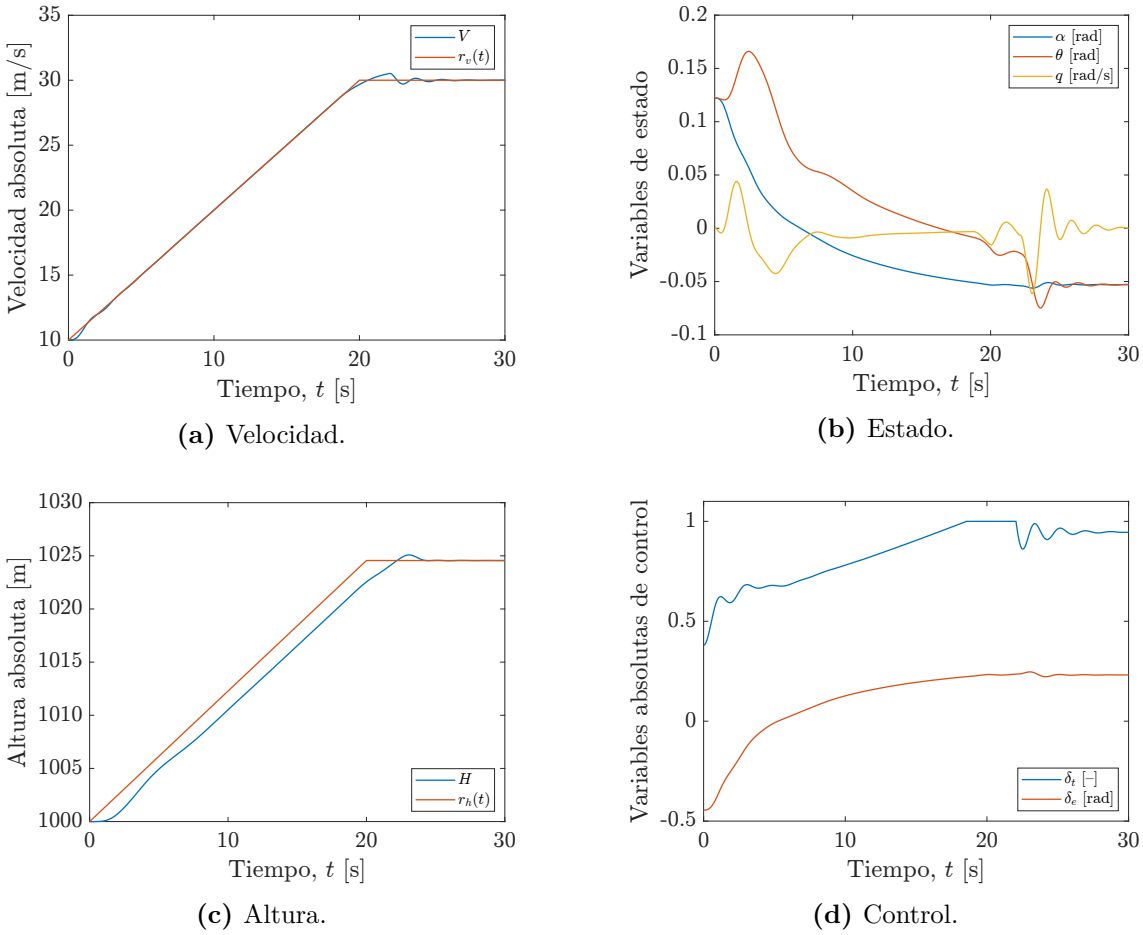
(d) Control.

Figura 4.14: Respuesta del UAV a un sistema de tracking de la velocidad ($r_v(t)$, figura (a)) y la altura ($r_h(t)$, figura (b)) empleando planificación de ganancias a partir de una familia de controladores definidos cada 5 m/s.

ja de oscilar en ningún momento, adquiriendo aceleraciones angulares más que notables (por las grandes pendientes de q). Por otro lado, las variables de control (y especialmente la posición del acelerador, pues es la variable menos penalizada; recuérdese la matriz de costes R (3.10)) también sufren oscilaciones elevadas a lo largo de esta maniobra.

Por otro lado, aprovechando esta pequeña familia de controladores diseñados para varias velocidades y alturas, se puede emplear la técnica de la **planificación de ganancias (*gain scheduling*)** para dar un paso más en la buena dirección. Este método, muy usado en aeronaves reales, consiste en ir realizando una interpolación entre las ganancias de esta familia de controladores en función del punto de operación en que se encuentre el avión (según unas *variables de planificación* que, en este caso, serían la pareja velocidad-altura). Con ello, implementando este método en Simulink y utilizando la misma familia que para la simulación anterior se obtiene la figura 4.14.

Se puede comprobar que, utilizando esta técnica, se mitigan todos los problemas de saltos de las variables en los cambios de intervalo pues se realiza una transición continua entre las ganancias de los controladores implicados. No obstante, aún puede observarse que, pese a que la evolución de los ángulos de ataque y de asiento es mucho más suave, el



(a) Velocidad. **(b)** Estado.

(c) Altura. **(d)** Control.

Figura 4.15: Respuesta del UAV a un sistema de tracking de la velocidad ($r_v(t)$, figura (a)) y la altura ($r_h(t)$, figura (b)) empleando en cada instante el controlador que proporciona la red neuronal.

intervalo de entre los 7 y los 20 segundos sigue presentando pequeñas irregularidades para la velocidad de cabeceo. Por supuesto, la relevancia de este hecho es incomparablemente menor al caso anterior. Cabe destacar también que, si se realiza una planificación de ganancias a partir de una familia de controladores que divida la envolvente de vuelo en intervalos más cortos para la velocidad (por ejemplo, cada metro por segundo en vez de cada 5), estas irregularidades se reducen aún más.

Y, por último, llegando al objetivo de la sección, se puede sustituir esta interpolación por parte de la planificación de ganancias por **la red neuronal**. Así, en cada instante de tiempo, la red traduce el estado del sistema (en este caso, la pareja velocidad y altura, que actúa como las anteriores «variables de planificación») a las 10 ganancias que debiera llevar el controlador en ese punto según el dataset con el que ha sido entrenada. La misma simulación empleando esta técnica proporciona la figura 4.15.

Se puede observar que la respuesta del sistema es la más suave de las obtenidas hasta ahora. El seguimiento de la rampa en velocidad y el consiguiente descenso del ángulo de ataque por su hipérbola son prácticamente perfectos; y la trayectoria de la altura es casi idéntica al caso del gain scheduling y más que ajustada a lo que cabría esperar de este

sistema de tracking (dados los coeficientes de costes definidos en su diseño, sección 3.6.3). Por su parte, la velocidad de cabeceo responde también de una manera más controlada que con los sistemas anteriores, y las variables de control hacen lo propio. Es con esta velocidad de cabeceo donde realmente se puede apreciar lo que se ha ganado al emplear la red neuronal.

Nótese, no obstante, que el inicio de la maniobra sí presenta algunas oscilaciones. Esto es debido a que, como se ilustraba en la figura 4.10, la red distaba ligeramente del controlador óptimo para bajas velocidades. Las oscilaciones del cambio de tramo a los 20 segundos, por su parte, son algo inherente a hacer el cambio de las referencias tan abrupto. Diseñando una transición menos repentina también se suavizaría la respuesta.

Todo lo necesario para obtener una idea general de cómo se han implementado los sistemas empleados en este apartado se detalla en el apéndice C.5. Dicha sección contiene e modelo principal y sus subsistemas y hace referencia a varios códigos de Matlab (apéndice A) y de Python (apéndice B).

4.6. Consideraciones sobre el diseño de la red neuronal

En las secciones anteriores se han mostrado los resultados que proporcionaba lo que hasta ahora se ha llamado simplemente «la red». Sin embargo, aún es posible analizar con algo más de detalle las implicaciones que tienen los hiperparámetros empleados en el diseño del modelo y la propia arquitectura de la red.

Recordando la sección 4.5, las principales características del *modelo original de red neuronal* eran las siguientes:

- **Arquitectura:** 4 capas ocultas de tamaños (20/40/40/20)
- **Funciones de activación:** Rectified Linear Unit (ReLU)
- **Función de coste:** Error Cuadrático Medio (MSE)
- ***Learning rate*:** 0.001
- ***Drop probability*:** 0.1
- **Algoritmo de optimización:** Método Adam
- **Número máximo de epochs:** 500
- ***Early stopping patience*:** 20
- ***Training batch size*:** 100

Y por otro lado pero de casi igual importancia, el dataset que se empleaba para entrenar a la red (que discretizaba el dominio de entre los 10 y los 30 m/s y hasta los 3100 metros de altura; y que de ahora en adelante será el *dataset A*) había sido el siguiente:

- **Puntos en velocidad:** 81 (cada 0,25 m/s)

- **Puntos en altura:** 31 (cada 100 m)
- **Puntos totales del dataset:** 2511 (malla de 81x31)

Por último, de todos estos puntos siempre se considerará un 80 % para la fase de entrenamiento y el 20 % restante para la etapa de test.

4.6.1. Arquitectura y dataset

En primer lugar, conservando el resto de parámetros comentados, se puede modificar la complejidad que tiene la red en cuanto a su estructura añadiendo y quitando capas ocultas. Esto, al igual que aumentar el tamaño de las propias capas (su número de neuronas), incrementa la cantidad de parámetros de la red y, con ello, tanto su capacidad de «aprender» como el coste de su entrenamiento. Por otro lado, la selección de puntos de la malla para obtener el dataset puede ser más o menos densa.

Con ello, para el siguiente análisis se considera el dataset A original presentado anteriormente y un nuevo *dataset B* más reducido y con las siguientes propiedades:

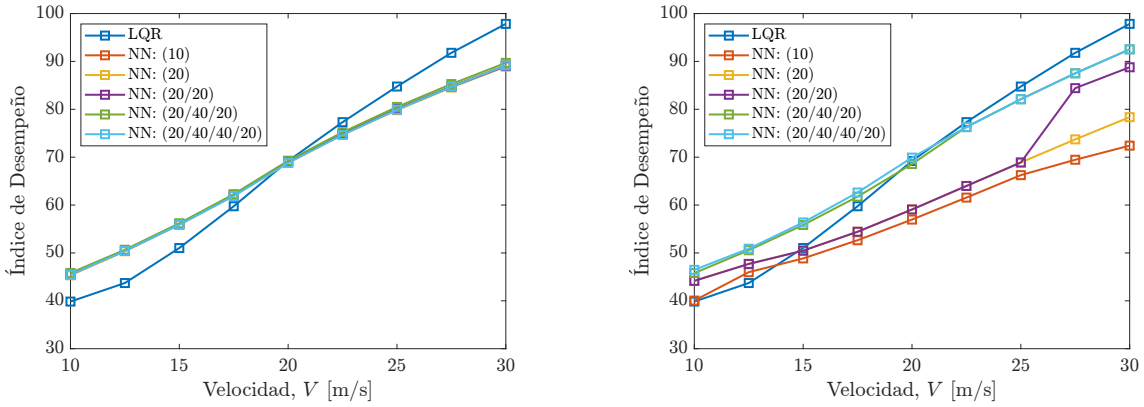
- **Puntos en velocidad:** 21 (cada 1 m/s)
- **Puntos en altura:** 7 (cada 500 m)
- **Puntos totales del dataset:** 147

Como se comentaba, se va a analizar cómo cambia el comportamiento al emplear distinto número y tamaño de capas ocultas. Para ello, se empelan las siguientes arquitecturas (cada número denota una capa oculta, separadas entre sí por barras), ordenadas con complejidad creciente: (10), (20), (20/20), (20/40/20), (20/40/40/20). Con todo ello, la figura 4.16 compara los PI (con la perturbación de referencia y para todo el rango de velocidades) que proporciona la casuística anterior[1]. Se muestra también el controlador por LQR de referencia.

Como se puede observar, para el caso de un dataset denso (figura 4.16a), incluso empleando únicamente una capa oculta con 10 neuronas, un modelo sencillo consigue replicar de una forma casi idéntica el resultado de la red más compleja analizada (20/40/40/20). Es decir, si el dataset contiene suficiente número de puntos, la red puede ser extremadamente reducida.

Sin embargo, con un dataset que en lugar de tener 2501 puntos (como el A), tenga tan solo 147 (dataset B), sí se encuentra que es necesaria cierta complejidad por parte de la red para compensar la escasez de puntos de entrenamiento del dataset (figura 4.16b). Esto se manifiesta con que las primeras tres redes ((10), (20) y (20/20)) presentan un comportamiento demasiado alejado del esperado. Este problema, como se comentaba, se soluciona en cuanto se pasa a tres capas ocultas (como con el caso de (20/40/20)) y

[1] Conviene mencionar que, debido al número de casos disponible para el entrenamiento (el 80 % del número total de puntos), para el entrenamiento de las redes que beben del dataset B (con 147 puntos totales) se ha reducido el *training batch size* a 40.



(a) Dataset A. **(b)** Dataset B.

Figura 4.16: Comparación del Índice de Desempeño (a 1000 metros de altura y en función de la velocidad) del controlador original (LQR) con el proporcionado por la red neuronal (NN) para distintas arquitecturas del modelo (entre paréntesis en las leyendas). La izquierda (a) emplea el dataset A y la derecha (b) el B.

apenas se consigue mejora respecto a esta última cuando se emplea una red con 4 capas como la (20/40/40/20). Nótese que, además, estas dos redes consiguen aproximarse más al comportamiento óptimo del controlador de referencia a altas velocidades que cualquiera de las redes entrenadas con el dataset A.

Por otro lado, dejando a un lado el Índice de Desempeño, en el proceso de test de la red ya entrenada se obtiene un valor de la función de coste (el error cuadrático medio) que, en principio, debiera dar información de cómo de buena es la red. Esto se recoge en la tabla 4.1. Se identifica cada dataset con su número de puntos totales, la arquitectura de la red, el número de parámetros de la misma y el mencionado error de evaluación (MSE). Además, la última columna de esta tabla incluye una medida de cómo de lejos del controlador por LQR resulta estar la red si se comparan sus PI. (Esto se ha calculado como la media de la distancia absoluta entre estas dos magnitudes para el rango de velocidades analizado: algo similar a la integral «discreta» del error entre la curva de la red considerada y la del LQR de la figura 4.16).

Así, como se concluía gráficamente en la figura 4.16, la tabla deja ver (tanto por MSE como por PI) que todas las redes entrenadas con el dataset A son prácticamente idénticas. Se puede ver también el mayor error de las tres primeras redes del dataset B y que la última de ellas consigue el menor error de todas las redes estudiadas. De esto se obtiene una conclusión importante, y es que: la medida de la calidad esperada de un controlador según el proceso de test de la red (el MSE) no replica exactamente cómo de bien logrará controlar al UAV (es decir, cómo de cerca estará su PI del controlador de referencia). El ejemplo claro de esto es la red (20/40/40/20) de dataset B que, con una función de coste mayor que cualquiera de las redes del dataset A, consigue un menor error de PI.

De todo lo anterior se puede concluir también que en este problema de Deep Learning compensa cargar más responsabilidad a la red y menos al dataset. Normalmente no se

Tabla 4.1: Comparación de la bondad de las distintas redes y datasets.

Dataset	Red	N. Parámetros	Coste (MSE)	Error (PI)
A (2511 puntos)	(10)	140	3.09E-04	4.8091
	(20)	270	1.27E-04	4.7938
	(20/20)	690	1.64E-04	4.7829
	(20/40/20)	1.9E3	2.44E-04	4.6284
	(20/40/40/20)	3.6E3	3.11E-04	4.7852
B (147 puntos)	(10)	140	2.49E-02	10.0812
	(20)	270	3.01E-02	5.6949
	(20/20)	690	5.87E-03	5.3256
	(20/40/20)	1.9E3	5.65E-03	4.9526
	(20/40/40/20)	3.6E3	7.60E-03	3.9990

tiene la posibilidad de *diseñar* el dataset. Sin embargo, este es un problema particular en el que sí se eligen y generan los puntos de entrenamiento y validación que interese (como se ha hecho empleando, por ejemplo, los dos tipos de datases anteriores). El problema está en que, pese a la ventaja de que se puede disponer del dataset que se desee, hay que asumir el coste computacional asociado a generarlo: y este proceso de obtención de la familia de controladores es bastante más costoso que el entrenamiento de cualquiera de las redes que se han estudiado. Esto es debido a que en la generación del dataset (diagrama de la figura 4.7) intervienen los procesos de trimado, linealización y obtención del controlador por LQR, que son procesos que Matlab realiza de manera más o menos lenta y que se tienen que repetir para un gran número de puntos. Por tanto, en cuanto a coste computacional y tiempo, conviene utilizar antes la combinación [Dataset B]-[Red (20/40/40/20)], que la pareja [Dataset A]-[Red (10)]. La demostración definitiva de esto se encuentra en el capítulo 5 donde, ampliando la complejidad del modelo (el número de variables de entrada) y reduciendo el número de puntos de entrenamiento por cada dimensión considerada (discretizando, por ejemplo, la velocidad cada 5 m/s y no cada 1 m/s), se consiguen resultados de calidades comparables a los aquí analizados empleando nuevas mallas considerablemente menos «densas».

4.6.2. Ajuste de hiperparámetros: *training batch size* y *drop probability*

Dos de los hiperparámetros presentados anteriormente y que merecen cierta discusión son el *training batch size* y la *drop probability*. El primero de ellos escoge la cantidad de casos que se introducen en la red de una misma vez en su proceso de entrenamiento y el segundo determina la probabilidad de que cada neurona de la red se desactive en el siguiente paso de entrenamiento.

Generalmente, la «probabilidad de drop» en los problemas de Deep Learning cuida

que la red no acabe «aprendiéndose» los datos de entrenamiento y sobreajustándolos demasiado (lo que se conoce como *overfitting*) y que luego no sea capaz de tener un buen desempeño en la etapa de test (pues se hace con nuevos datos distintos a los del entrenamiento de la red) o cuando el modelo sea puesto en producción (o en el caso de nuestro UAV, en vuelo). Así, una probabilidad más alta trata de prevenir en mayor medida este efecto.

Para este problema en particular, ya que se trata de sustituir el tradicional método de planificación de ganancias por un interpolador de controladores, lo curioso es que sí es de interés que el modelo acabe *aprendiéndose* los puntos con los que se entrena. Esto es debido a que, efectivamente, cuando el sistema realmente estuviera en vuelo y siempre que no se dieran circunstancias extraordinarias (como alcanzar un punto de operación fuera de la envolvente de vuelo), los únicos estados que vería la red serían muy próximos a los puntos de entrenamiento. Conviene remarcar que esto es realmente aplicable cuando el mallado de la envolvente de vuelo es lo suficientemente denso. Es decir, esto es más verdad para el caso de las redes que beben del dataset A (que discretizaba la velocidad cada 0,25 m/s) que para las del dataset B (que lo hacía cada 1 m/s)[2].

Por otro lado, el *training batch size*, el tamaño de los conjuntos de puntos de entrenamiento, tiene que ver con factores como la velocidad de entrenamiento. Por lo general, un mayor batch size aligera el tiempo de ejecución de cada epoch (iteración completa en la que todo el conjunto de datos de entrenamiento ha pasado por la red). Además, este parámetro influye en la diversidad de puntos que ve la red en cada iteración. Un mayor batch size hace que puedan introducirse a la red una mayor variedad de puntos de entrenamiento en cada paso del optimizador y puede tener implicaciones en cómo de bien funciona el modelo. Un número muy reducido tampoco suele ser interesante porque la red trataría de ajustar sus pesos y bias (los parámetros a optimizar de cada neurona) para replicar las ganancias de los puntos de operación particulares que se hayan introducido, y esto puede llevar asociados problemas de convergencia y un número de epochs demasiado alto.

Con ello, se pueden variar estos hiperparámetros para ver cómo se desenvuelve la red. Así, la tabla 4.2 lo hace para el dataset A y la tabla 4.3 considera el dataset B. En la cabecera de las tablas, «p(drop)» indica el valor de esta probabilidad de hacer drop de cada neurona y, en la dimensión vertical, *train batch size* es abreviado como «tbs». Además, para cada caso, «Epochs» recoge el número de iteraciones de la red que hicieron falta para legar al criterio de parada de paciencia 20[3]; «t[s]» indica el tiempo de entrenamiento que requirió la red; y «MSE» es el error cuadrático medio de la red en la etapa de validación con el ya mencionado 20 % de los datos. Para esta comparación, según lo discutido en la sección anterior, se va a usar como red la de arquitectura (20/40/20), pues ya obtenía

[2]La partición en altura, como se ha ido viendo, no es tan significativa en estos aspectos, pues la variación de velocidad modifica en mucha mayor medida la dinámica del sistema que un cambio en la densidad atmosférica derivada de una altura distinta.

[3]Este valor significa que el proceso de entrenamiento acaba antes de llegar al número máximo de epochs: se detiene cuando trascurren 20 iteraciones sin que haya habido una mejora en la función de coste.

Tabla 4.2: Hiperparámetros con el dataset A.

	p(drop) = 0.1			**p(drop) = 0.2**			**p(drop) = 0.3**		
tbs	Epochs	t[s]	MSE	Epochs	t[s]	MSE	Epochs	t[s]	MSE
50	157	33	8.90E-04	133	27	3.39E-03	119	27	3.02E-03
100	217	30	3.20E-04	183	33	1.79E-03	163	23	2.23E-03
200	394	31	5.23E-04	313	43	9.57E-04	266	29	1.84E-03
400	350	24	6.66E-04	493	40	7.80E-04	394	31	1.63E-03
800	271	13	7.53E-04	299	18	1.48E-03	612	50	1.28E-03
1004	234	11	1.72E-03	239	14	2.21E-03	492	28	3.32E-03

Tabla 4.3: Hiperparámetros con el dataset B.

	p(drop) = 0.1			**p(drop) = 0.2**			**p(drop) = 0.3**		
tbs	Epochs	t[s]	MSE	Epochs	t[s]	MSE	Epochs	t[s]	MSE
10	193	11	1.19E-03	188	12	1.63E-03	226	15	5.74E-03
20	79	4	7.16E-03	110	5	2.95E-03	167	8	4.62E-03
30	90	5	2.94E-03	90	5	4.70E-03	108	5	8.92E-03
40	128	4	3.47E-03	122	6	5.52E-03	212	8	5.62E-03
58	134	6	8.49E-03	73	3	2.65E-02	160	6	1.79E-02

resultados apropiados para ambos datasets. Del mismo modo, la evaluación del error de la fase de test (MSE), se va a emplear siempre un *test batch size* (de significado idéntico al *train* salvo que para la fase de evaluación) igual a 20, independientemente del valor que tome en el entrenamiento y del tamaño del dataset. Esto es para intentar asegurar la uniformidad en los resultados presentados.

En primer lugar, observando el sector con probabilidad de drop 0,1 de la primera tabla, se puede apreciar que un mayor tbs implica un menor tiempo de entrenamiento. Esto no ocurre, sin embargo, con el número de iteraciones: un mayor tbs no ha llevado asociado un descenso de este número de epochs de parada. Observando los mismos valores pero para el caso del segundo dataset (tabla 4.3), la tendencia para el número de epochs y el tiempo de entrenamiento parece encontrar un mínimo en $tbs = 20$, aunque las diferencias entre los valores que toman estos dos campos son significativamente menos notables que para el caso del dataset A. El tiempo de entrenamiento oscila en torno a valores cercanos a los 5 segundos y puede ser debido a procesos paralelos que estuvieran ocurriendo en el ordenador donde se ejecutaba el proceso. Por supuesto, si se comparan los valores absolutos del tiempo de entrenamiento, el dataset B es mucho más rápido que el A debido al incomparable número de puntos (147 frente a 2511).

Considerando el MSE de la red, el caso del dataset A permite observar que una mayor probabilidad de drop hace que aumente este índice de error de evaluación. Obsérvese que con una probabilidad de 0,1 todos los MSE son del orden de la diezmilésima, para pasar a

oscilar entre este valor y diez veces más cuando se usa 0,2 de probabilidad, y alcanzar el MSE del orden de las milésimas para una probabilidad de 0,3. Esto, como se comentaba antes, es porque realmente interesa que la red «se aprenda» los datos de entrenamiento, porque son los que posteriormente se va a encontrar en su vida útil (y aquí esta vida útil ha sido personificada con el conjunto de valores de test). Obsérvese, no obstante, el caso de $tbs = 1004$ (la mitad del tamaño de los datos de entrenamiento del dataset A, 80 % de $2511 = 2008{,}8$). En este caso, la red no se comporta tan bien como en los anteriores y su error es ligeramente mayor. Este efecto de un mayor MSE a mayor p(drop) no se observa en el caso del dataset B. Puede volver a verse que para $tbs = 58$ (la mitad en este caso) el error es mayor. Además, el efecto del tbs en el tiempo y el número de epochs ha sido menos claro que en para el dataset A.

No obstante, por mucho que se estén comparando los errores de las distintas combinaciones de hiperparámetros, conviene recordar que una red con un MSE del orden de incluso las centésimas (como los que se recogían para las redes más sencillas de la anterior tabla 4.1) proporciona unas ganancias que consiguen controlar de manera más que satisfactoria el movimiento del UAV. Estas comparaciones pretenden poner de manifiesto la naturaleza del estudio (considerando, por ejemplo, el caso particular de la probabilidad de drop de este problema) o ciertas consideraciones generales. No es el objetivo de este estudio obtener una red perfecta que minimice al máximo el error ya que se ha visto que incluso una red sencilla (sección 4.6.1) consigue resultados satisfactorios y que un menor error de validación no implica necesariamente un mejor PI.

Por otro lado, para un análisis en más detalle haría falta ejecutar varias veces cada modelo y hacer una media de los resultados obtenidos. Esto es debido a que los valores de (epochs, tiempo y MSE) de una red particular dependen de cuándo se ejecute: pese a que la inicialización de los parámetros de la red se haga por defecto al definir el proceso de entrenamiento (listing B.1), cada vez que se realiza la partición aleatoria el 80 % del dataset para el entrenamiento se obtienen unos valores distintos, por lo que el «contexto» de entrenamiento de la red es distinto cada vez. De todas formas, aunque se definiera una forma determinista de dividir el dataset que contuviera una diversidad de parámetros suficiente, existe algún proceso aleatorio dentro de la inicialización de los parámetros de la red o en el algoritmo de optimización que, incluso usando los mismo puntos del dataset, hace que dos entrenamientos consecutivos de la red generen resultados (de número de epochs, tiempo, y MSE) ligeramente distintos. Este comportamiento se podría cambiar modificando el parámetro `deterministic = True` del entrenamiento (`pl.Train`) de la red (listing B.1).

4.6.3. Función de activación

La función de activación empleada en todas las capas de las redes estudiadas hasta ahora era la «Rectified Linear Unit» (ReLU), que se define como:

$$\mathrm{ReLU}(x) = \max(0, x) \tag{4.5}$$

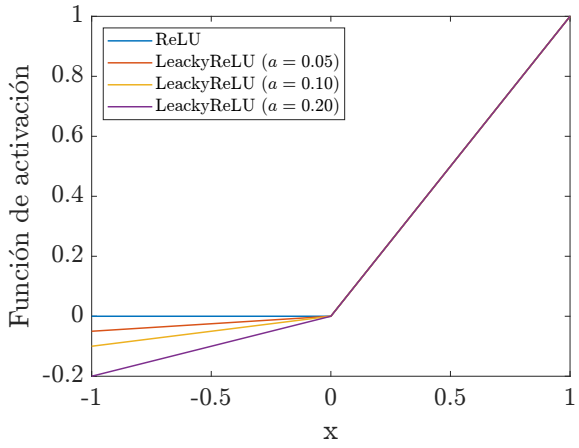


Figura 4.17: Distintas funciones de activación de la red neuronal.

Esta función de activación es una de las más sencillas que permiten recoger comportamientos no lineales de los datos de aprendizaje y es usada en muchas aplicaciones. Sin embargo, esta función tiene asociado el fenómeno de que las neuronas «mueran» cuando sus entradas toman valores negativos (pues su salida es nula).

En nuestro problema en concreto, las variables de entrada, pese a ser valores positivos (velocidad y altura), son normalizadas antes de entrenar a la red (asignándoles valores según una distribución normal estándar), por lo que la primera capa de neuronas sí ve valores negativos en ciertos casos y estaría «muriéndose». Pasada la primera capa, no obstante, como todas las funciones de activación devuelven valores positivos o nulos, podría esperarse que no se diera este efecto. Sin embargo, habría que analizar cómo afectan los bias de cada neurona a que los argumentos de entrada de las ReLU sean positivos o negativos, y lo mismo habría que hacer con los pesos de las mismas.

En cualquier caso, esto que se puede estar enfocando como un problema realmente no lo es. Que el mecanismo de la red para aprender comportamientos no lineales del sistema sea haciendo morir a ciertas neuronas no es un problema que debiera molestarnos.

Aún así, tratando de encontrar soluciones alternativas, se puede usar la función «**Leaky ReLU**» que incorpora una pendiente a para los argumentos menores que cero:

$$\text{LeakyReLU}(x) = \max(ax, x) \tag{4.6}$$

donde esta a suele tomar valores entre 0,01 y 0,2.

La diferencia entre esta LeackyReLU y la ReLU original se muestra en la figura 4.17. Está función suaviza el efecto de muerte de las neuronas y obtiene un mejor gradiente para el optimizador en la *backpropagation* a costa de perder cierta no-linealidad para pendientes elevadas.

Pese a estas diferencias teóricas en el comportamiento de las distintas funciones, los resultados de incorporar esta nueva función de activación en las capas de la red carecen de peculiaridades de interés. El desempeño de la red neuronal es prácticamente idéntico

que con las ReLU originales y no se ha observado ninguna otra diferencia significativa.

4.6.4. *Learning rate*

El ritmo de aprendizaje (*learning rate*) es otro parámetro del proceso de entrenamiento que tiene que ver con cómo de pequeños son los pasos que da el algoritmo de optimización en el proceso de propagación para atrás (*backpropagation*). Este algoritmo, como se introdujo cuando se presentó la red, se trata del método Adam: un algoritmo de primer orden basado en el gradiente muy usado para funciones objetivo estocásticas[4].

La variación de este parámetro suele tener implicaciones sobre la velocidad de entrenamiento y el número de epochs asociado pero también con la capacidad de convergencia del método, dependiendo de cómo se comporte la función de costes internamente a la actualización de los parámetros de la red.

Con objeto de estudiar las implicaciones que varios learning rates (lr) pudieran tener en nuestro entrenamiento de la red, se vuelve a considerar la arquitectura (20/40/20) para el caso de los datasets A y B. Además, como se comentaba en apartado 4.6.2, los resultados presentados para cada caso se corresponden a un promedio de tres entrenamientos sucesivos (reseteando la red, por supuesto), con el objetivo de intentar recoger de una forma más fiel las cifras de número de epochs y tiempo de entrenamiento. La función de coste de test (MSE) no se ve tan afectada entre los distintos entrenamientos pero, aún así, los valores mostrados también se corresponden a dicho promedio. Por otro lado, los valores de *train batch size* que se han usado para los datasets A y B son, respectivamente, 400 y 30, eligiendo un valor que represente una fracción similar del número total de puntos de entrenamiento en cada caso. La tabla 4.4 recoge todo lo anterior.

Tabla 4.4: Variación del ritmo de aprendizaje (lr).

	Promedio con el dataset A			**Promedio con el dataset B**		
lr	Epoch	t [s]	MSE	Epochs	t [s]	MSE
1E-01	45	3	4.26E-02	65	2	1.30E-02
5E-02	117	9	1.73E-03	95	3	2.66E-03
1E-02	306	21	9.00E-04	98	3	2.92E-03
5E-03	574	40	5.07E-04	172	6	2.57E-03
1E-03	360	24	1.89E-03	151	5	5.48E-03
5E-04	275	16	3.33E-03	189	6	5.13E-03

En primer lugar, observando la tabla se puede comprobar que un $lr = 0{,}05$ ya es suficiente para conseguir un error del orden de las milésimas en ambos casos. Se encuentra también que un lr menor suele implicar un aumento en el tiempo de entrenamiento, siendo más significativo el efecto en el caso del dataset A. Algo notable se encuentra en la zona central de la tabla (siendo de nuevo más apreciable para el dataset A) cuando se usan

[4]Para más información sobre el algoritmo puede consultarse el artículo de Kingma and Ba (2014).

lr intermedios. Se puede ver que con $lr = 0{,}005$ se alcanza el menor error de todos con el dataset A, requiriendo por otro lado el mayor tiempo promedio de entrenamiento y número de epochs. El anterior $lr = 0{,}01$ logra también un MSE del orden de 10 a la menos 4 con menos tiempo y epochs, aunque se encuentra cerca en error del caso del segundo $lr = 0{,}05$, que requiere menos de la mitad de epochs y tiempo.

Para el caso del dataset B, se encuentra que el mínimo error se corresponde también con el lr = 5E-3. Y, como se comentaba, las implicaciones que tiene este análisis en el tiempo y número de epochs es menos significativo, llegando a encontrar fenómenos particulares (como que el máximo de epochs no se encuentre en el centro de la tabla, sino al final).

Como siempre, estos análisis tratan de obtener ideas generales sobre el comportamiento del proceso de entrenamiento y el objetivo no es sacar conclusiones definitivas. Esto se ve claramente con el caso de los tiempos empleados para el dataset B que, pese a ser distintos, las diferencias que se encuentran no tienen ninguna implicación práctica de provecho. Del mismo modo, por más que se haya dicho que hay diferencias entre los tiempos empleados para el dataset A, estos son también todos «iguales» a efectos prácticos.

Como conclusión sí se puede remarcar que el menor error obtenido se produce para un ritmo de aprendizaje del orden de $lr = 0{,}005$ y que éste consigue un MSE de 10 a la menos 4. Este resultado confirma los valores recogidos anteriormente en la tabla 4.2 y fija que dicho MSE representa el mejor desempeño que se ha conseguido para la red neuronal en todo este estudio.

Capítulo 5

Ampliación del modelo a más dimensiones y mejoras generales

5.1. Introducción

Hasta ahora, el modelo de red neuronal que se ha empleado ha considerado como entradas la velocidad y la altura del UAV en su condición de vuelo instantánea y se ha comprobado que proporciona resultados más que comparables (si no mejores) a los de la planificación de ganancias.

Este capítulo trata, en primer lugar, de ampliar la complejidad del modelo. Aprovechando las propiedades de las redes neuronales, se pueden considerar más variables de entrada con simplemente modificar la primera capa de la red y ajustar la arquitectura de la misma en caso de que fuera necesario aumentar el número de parámetros de las neuronas. Con ello, se recuperan los tres parámetros geométricos y másicos que se introdujeron en la sección 2.3.4 y que han ido apareciendo en las primeras líneas de todos los códigos empleados hasta el momento. Estos parámetros son: la masa del UAV, la posición longitudinal de su centro de masas y su momento de inercia de cabeceo. Este modelo de red permitiría al controlador adaptarse a las ganancias más adecuadas, además de en cada punto de su envolvente, cuando la aeronave experimentase cambios en sus condiciones másicas. Esto podría simular la evolución del combustible en un vuelo real o el caso de un avión militar o carguero que estuviera desalojando una gran cantidad de masa en forma de mercancías.

Con ello, al apartado 5.2 analiza cómo se ha modificado el mallado de la envolvente de vuelo más estas tres nuevas dimensiones para generar el dataset correspondiente. El siguiente, 5.3, muestra cómo de bien se comporta la nueva red *pentadimensional* en la fase de test y la emplea en el problema de tracking para unos valores de los parámetros concretos.

Por otro lado, el aparatado 5.4 indaga en los efectos que tiene reducir la complejidad del dataset y encuentra que incluso muy pocos puntos son suficientes para entrenar a la red. Por último, aprovechando las implicaciones de tener un dataset reducido, el apartado

hace una última simulación del problema de seguimiento con un tamaño del dataset que trate de replicar una situación de diseño real.

5.2. Ampliación del dataset a cinco dimensiones

El dataset a generar para el entrenamiento de la red se modifica de diversas formas. En primer lugar, las **velocidades** consideradas se extienden en el intervalo de 10 a 35 m/s (en lugar de hasta los 30 m/s como en los casos anteriores). Si se recuerda la envolvente de vuelo del UAV inicial (es decir, con los parámetros másicos de referencia), figura 2.8, la frontera de altas velocidades por insuficiencia de empuje estaba alrededor de los 31 m/s. Esto implica que ampliando hasta 35 m/s el dataset estamos considerando puntos de operación en los que el UAV no puede mantener el vuelo horizontal rectilíneo y uniforme. Aun así, el motivo de hacerlo es por considerar un rango más amplio de velocidades (y de presiones dinámicas) que varíen más la dinámica del sistema, con el objetivo de aumentar el rango de condiciones que tiene que aprender la red. Dicho esto, la generación de los controladores en los puntos fuera de la envolvente de vuelo se hará de la misma forma con el método LQR (igual que en los mapas de color del apartado 4.2).

En cuanto a la **altura**, se extiende el rango de operación hasta los 6000 metros. Esto no entra en conflicto con la frontera superior de la envolvente de vuelo pues, como se indicó en su análisis, el techo teórico real de la aeronave se encuentra mucho más arriba. De nuevo, este salto de los 3 a los 6 kilómetros de rango es para hacer más importantes las variaciones de densidad y que la dinámica de la aeronave se modifique en mayor medida debido a una variación de este parámetro de altura[1].

Con las dimensiones de los tres nuevos **parámetros másicos y geométricos** del UAV se van a considerar variaciones del 25 % respecto a los valores de referencia empleados hasta ahora. Para estos nuevos puntos no se realiza ningún análisis de si pertenecen o no a una hipotética envolvente de vuelo modificada pues carece de interés. Como para el caso de la ampliación hacia mayores velocidades, no se quiere estudiar concretamente si los puntos considerados permiten o no el vuelo. Una posición más adelantada del centro de masas, por ejemplo, cambia el coeficiente de momento y modifica la deflexión del timón necesaria para equilibrar el avión. De la misma forma, una mayor masa aumenta el ángulo de ataque necesario para que la sustentación compense al peso y la frontera izquierda de la envolvente se vería afectada. Sin embargo, pese a estos ejemplos y a múltiples otras implicaciones, como se mencionaba, se considerarán dichas variaciones del 25 % para estos parámetros con el único objetivo de conseguir una mayor variabilidad del sistema, sin importar cómo se comportase el UAV en cada una de estas nuevas condiciones. Se diseñará el controlador correspondiente en cada punto con la técnica del LQR como se ha hecho hasta ahora.

Por tanto, para esta nueva generación del dataset se procedería como en el diagrama de la figura 4.7 empleando los cinco valores de las variables discutidas en el primero de los

[1] Siguiendo el modelo de la Atmósfera Estándar Internacional, la densidad a 6 km de altura es de 0.6595 kg/m^3, alrededor de la mitad que al nivel del mar. La densidad a 3 km es de 0.9090 kg/m^3.

bloques.

Asimismo, como ya se ha discutido, tras definir un dominio de operación es necesaria una discretización del mismo en puntos concretos para poder trimar y linealizar el listema y diseñar el controlador correspondiente. Para esta ampliación se está considerando un espacio de cinco dimensiones de variación de los parámetros: se tienen los dos originales (V y H) y los tres nuevos (W, x_{cg} e I_p), siguiendo la notación empleada en el modelo C.1. Y el problema reside en la decisión de cómo de densa hacer la malla. A modo de ejemplo, si se considera la discretización de las variables originales del dataset B (21 puntos en velocidad y 7 en altura), que era el «pequeño» de los dos datsets analizados hasta el momento, y se toman únicamente 5 puntos para las otras tres dimensiones, el dataset a generar tendría alrededor de 18000 puntos; y esto tendría un conste computacional muy alto.

Se podría argumentar que esta sería una tarea perfectamente realizable en un tiempo del orden de las horas. Sin embargo, sería un proceso exageradamente injustificado. Como se comprobará en los siguientes apartados, no son necesarios demasiados puntos por dimensión para obtener una red casi tan «perfecta» como la que se obtenía con los datasets A y B (sección 4.6). Con esto en mente, el dataset que se va a generar para este nuevo estudio *pentadimensional* y al que nos referiremos como *dataset V* es:

- **Puntos en velocidad:** 6 (rango = [10, 35] m/s, cada 5 m/s)
- **Puntos en altura:** 6 (rango = [0, 6000] m, cada 1200 m)
- **Puntos en masa:** 5 (rango = $[0,75W^*,\ 1,25W^*]$, $W^* = 2{,}5\,\text{kg}$)
- **Puntos en centro de masas:** 5 (rango = $[0,75\hat{x}^*_{cg},\ 1,25\hat{x}^*_{cg}]$, $\hat{x}^*_{cg} = 0{,}33$)
- **Puntos en inercia:** 5 (rango = $[0,75I^*_p,\ 1,25I^*_p]$, $I^*_p = 0{,}1568\,\text{kg m}^2$)
- **Puntos totales del dataset:** 4500 (malla de 6x6x5x5x5)

Por tanto, lo que se está haciendo es dividir la velocidad y la altura en 5 intervalos y el resto de parámetros en 4.

Conviene destacar que esto tiene implicaciones también sobre lo comentado en la comparación de los datasets A y B. Si lo que se dice sobre el buen funcionamiento de este dataset (con un número tan reducido de puntos por dimensión) es cierto (apartado siguiente), significaría que incluso el dataset B estaba sobredimensionado y que no era necesaria una discretización tan fina del entorno de vuelo: el dataset B tomaba puntos cada 1 m/s y este nuevo lo hace cada 5. Esto, sobre todo, se debe a que este es un problema más complejo que el anterior: la red ha de aprender a manejar entradas en cinco dimensiones, en lugar de las dos originales. Por tanto, si la red consigue funcionar adecuadamente con un problema más complejo y un dataset menos denso, el dataset B generado para el problema original más sencillo estaba, sin lugar a dudas, muy sobredimensionado.

5.3. Resultados de las redes neuronales en cinco dimensiones

Así, a partir del nuevo dataset generado según las consideraciones del apartado anterior se pueden entrenar distintas arquitecturas de red neuronal y combinar los hiperparámetros para intentar obtener el mejor resultado posible. Con este objetivo, se omiten ciertos pasos según los resultados del apartado del capítulo anterior: se fija el *learning rate* en $lr = 0{,}005$, el *training batch size* en $tbs = 200$ (ambos elegidos tras cierta prueba y error aunque sin demasiada influencia) y para el resto de parámetros se usan los de la red original (paciencia de 20 y ReLU como función de activación). El criterio de parada será, como siempre, el asociado a esta paciencia: acababa el entrenamiento si no se mejoraba la función objetivo en las últimas 20 iteraciones.

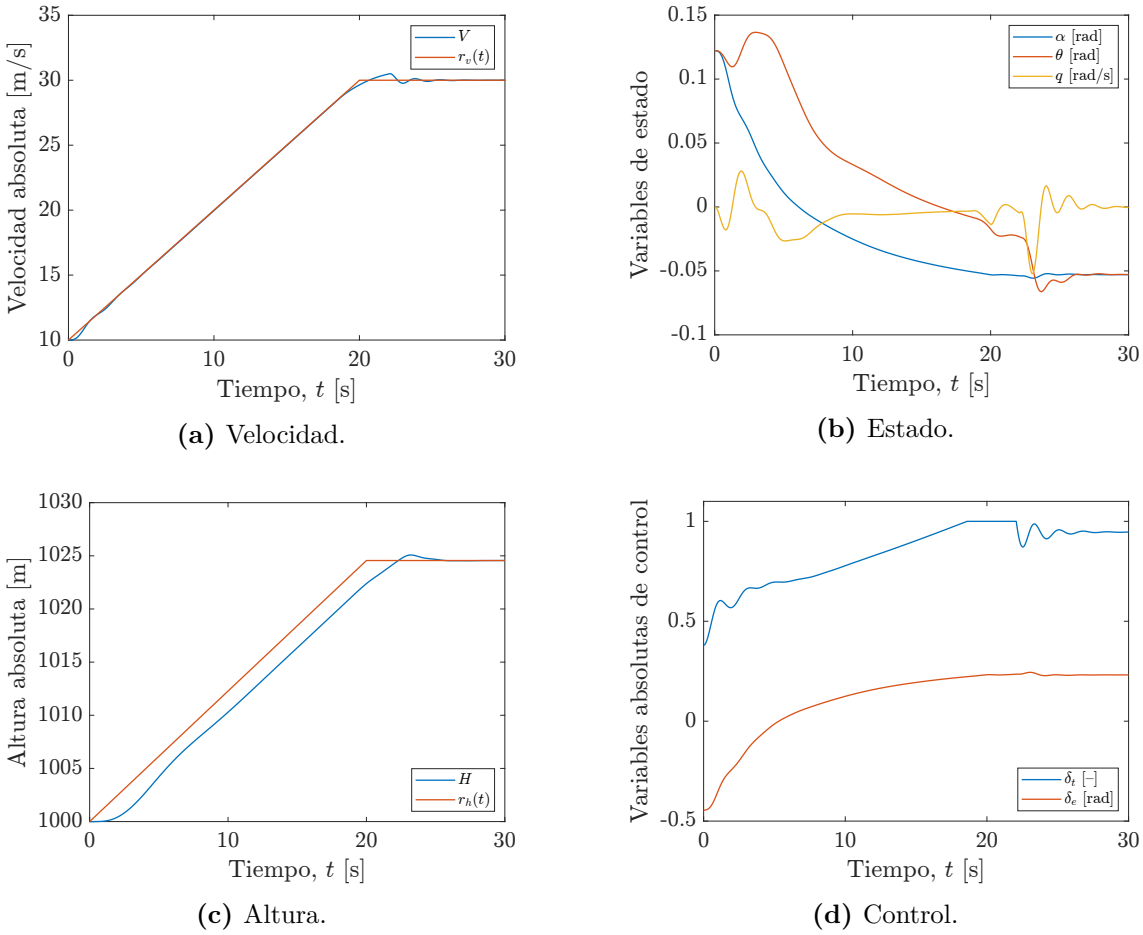
Consecuentemente, la tabla 5.1 compara los resultados de número de epochs de parada (campo «Epochs») con dicha paciencia del *early stopping* y de la función de coste de test (campo «MSE») para distintas arquitecturas (usando la notación habitual en el campo «Red»). A su vez, la tabla se divide en columnas según la probabilidad de drop que se haya empleado. Es este caso, además, se analiza también el caso de $p = 0$.

Tabla 5.1: Comparación de las distintas arquitecturas de red y su probabilidad de drop.

p(drop)	**p = 0**		**p = 0.1**		**p = 0.2**		**p = 0.3**	
Red	Epoch	MSE	Epoch	MSE	Epoch	MSE	Epoch	MSE
(10)	410	1.3E-3	263	8.7E-3	192	1.9E-2	226	3.0E-2
(20)	470	3.3E-4	253	4.8E-3	218	9.9E-3	197	1.5E-2
(20/20)	337	1.4E-4	227	1.9E-3	265	8.2E-3	230	1.5E-2
(20/40/20)	240	1.0E-4	286	3.6E-3	247	1.3E-2	200	2.1E-2
(20/40/40/20)	250	2.3E-4	260	6.2E-3	225	1.3E-2	187	2.4E-2

Observando los valores de tabla, en primer lugar, se puede despreciar la relevancia de las variaciones del número de epochs en cada caso, pues las diferencias no son significativas y el tiempo de entrenamiento tampoco se ve muy modificado (Es por ello por lo que se ha prescindido de dicho campo en la tabla, pues sigue la tendencia de análisis anteriores: a mayor número de parámetros, por lo general, mayor tiempo de entrenamiento, aunque como se indicaba, las diferencias son mínimas).

Con respecto al MSE sí se encuentran resultados significativos. Como ya ocurría en las tablas 4.2 y 4.3 con la red original bidimensional, aumentar la probabilidad de drop solo empeora los resultados: crece el error entre las ganancias obtenidas por la red y las del controlador original de referencia (que es lo que significa el MSE de test). Además, la introducción del caso con probabilidad nula (es decir, que no incorpora el mecanismo de drop y todas las neuronas de la red están activas en todo momento) hace que aparezcan los errores más pequeños de la tabla (del orden de la diezmilésima). Esto puede verse en



(a) Velocidad. **(b)** Estado.

(c) Altura. **(d)** Control.

Figura 5.1: Respuesta del UAV a un sistema de tracking de la velocidad ($r_v(t)$, figura (a)) y la altura ($r_h(t)$, figura (b)) empleando en cada instante el controlador que proporciona la red neuronal con peor MSE: (10) con $p = 0{,}3$.

que incluso la red más simple de 10 neuronas sin mecanismo de drop obtiene un MSE menor que la red más compleja (20/40/40/20) con $p = 0{,}1$.

De esto se vuelve a confirmar lo discutido en el apartado 4.6.2: en este problema de Deep Learning, lo mejor es no usar el mecanismo de drop. En dicho apartado se encontraba que una menor probabilidad mejoraba los resultados de la red. Aquí se ha analizado como nuevo caso la situación límite de tener probabilidad nula y se ha llegado a las mismas conclusiones.

Por otro lado, como siempre, cualquiera de estos errores (por mucho que difieran incluso en dos órdenes de magnitud) sigue dotando al UAV de una respuesta más que adecuada. Se puede tomar como ejemplo el caso de la red (10) con $p = 0{,}3$, que representa el mayor MSE de la tabla 5.1. La figura 5.1 muestra la respuesta del UAV al problema de tracking que se discutía en la sección 4.5 pero instalando esta nueva «peor» red (con más error en sus ganancias). Para esta simulación se ha tomado un UAV cuyo vector de parámetros es el siguiente:

$$\left[W, \quad \hat{x}_{cg}, \quad I_p\right] = \left[2{,}7 \text{ kg}, \quad 0{,}38, \quad 0{,}16 \text{ kg m}^2\right] \tag{5.1}$$

La elección de estos parámetros se debe a intentar escoger un punto de operación «alejado» de lo que haya podido ver la red en su entrenamiento. Así, estos puntos se encuentran entre medias de los nodos que emplea el dataset en cada una de estas dimensiones.

Como se puede comprobar, las diferencias entre esta nueva respuesta y la que se encontraba en la figura 4.15 son prácticamente inexistentes. La red, pese a haber dado un mayor error en la etapa de test, consigue controlar de manera satisfactoria las oscilaciones en el ángulo de cabeceo (que era el criterio que se seguía entonces para evaluar cómo de buena era un respuesta). Conviene hacer notar que los valores numéricos encontrados en los ejes de ordenadas de las gráficas (especialmente la relativa a las variables de estado, 5.1) son distintos a los de los anteriores seguimientos porque la aeronave es ligeramente distinta (por dicho vector de parámetros).

5.4. Análisis de los efectos de la simplificación del dataset

Recuperando una discusión anterior, los resultados de MSE de la tabla 5.1 (generada con el dataset V de 5 dimensiones) son del mismo orden que los que conseguían las redes con el dataset B de dos dimensiones (tabla 4.3). Esto confirma que, efectivamente, este último estaba sobredimensionado. Debido a esto, se quiere ver qué ocurre si se utiliza un dataset mucho menos denso que el dataset B para entrenar a la red bidimensional.

Un punto de partida conocido sería emplear la misma discretización de velocidad y altura que el dataset V para la red 5D, con 6 puntos en cada dimensión. Esto es equivalente a quedarnos con las dos primeras dimensiones del dataset V, particularizadas en los puntos centrales de las otras tres dimensiones (que fijarían los parámetros geométricos y másicos en sus valores de referencia). Sea de esta forma o ejecutando de nuevo el bucle de la figura 4.7, se obtiene el *dataset S* siguiente:

- **Puntos en velocidad:** 6 → [10, 15, 20, 25, 30, 35] m/s
- **Puntos en altura:** 6 → [0, 1200, 2400, 3600, 4800, 6000] m
- **Puntos totales del dataset:** 36 (malla de 6x6)

Y, llegando casi al extremo, el *dataset XS* considerará únicamente:

- **Puntos en velocidad:** 3 → [10, 22.5, 35] m/s
- **Puntos en altura:** 3 → [0, 3000, 6000] m
- **Puntos totales del dataset:** 9 (malla de 3x3)

Nótese que los rangos de velocidad y altura de estos dos nuevos datasets se extienden hasta mayores valores (los datasets bidimensionales A y B solo llegaban a velocidades de 30 m/s y a 3100 metros de altura).

Así, aprovechando también lo descubierto en el apartado anterior de que no usar el mecanismo de drop es la mejor técnica a seguir, se realiza la ya conocida comparación entre

Tabla 5.2: Comparación de los resultados de varias arquitecturas de red según el dataset utilizado.

	Red 2D: Datasets			
Propiedades	**A**	**B**	**S**	**XS**
Train samples	2008	117	28	7
Test samples	503	30	8	2
bs: train	400	30	7	4
bs: test	20	20	8	2
Red	MSE			
(10)	1.29E-04	1.41E-04	8.47E-04	2.76E-03
(20)	3.69E-05	1.35E-04	3.17E-04	3.19E-02
(20/20)	4.40E-05	1.17E-04	4.12E-04	9.81E-02
(20/40/20)	3.45E-05	5.08E-05	4.55E-04	7.08E-02
(20/40/40/20)	4.56E-05	9.78E-05	1.81E-03	3.95E-02

las arquitecturas de la red para todos los datasets considerados. Este análisis introduce los nuevos datasets S y XS y presenta cómo se comportan A y B con una probabilidad de drop nula (pues hasta ahora la mínima que se había usado con ellos era 0,1). Con ello, la tabla 5.2 recoge el número de puntos de los conjuntos de entrenamiento («Train samples») y de test («Test samples») y especifica el tamaño del *batch size* (*bs*) que se ha elegido para ambos procesos[2]. En la parte inferior de la tabla se encentra el ya conocido error MSE de test para las distintas arquitecturas estudiadas.

En primer lugar, como era de esperar, se comprueba que las redes entrenadas con los datasets A y B mejoran sus resultados respecto a los que obtenían con una pequeña probabilidad de drop (tablas 4.2 y 4.3): el dataset A, que dividía la velocidad en intervalos de 0,25 m/s, alcanza errores de 10E−5, los menores encontrados en todo el estudio. En las columnas de estos dos datasets también se puede apreciar como una mayor complejidad de red logra una disminución del error.

Los resultados del nuevo dataset S, del orden de la diezmilésima, confirman las anteriores sospechas relativas al dimensionado del dataset B. Recuérdese que, además, este dataset S abarca el doble de rango en alturas y algo más también en velocidad. Por tanto, se puede concluir que generar una discretización de todo el rango de velocidades en 5 intervalos (lo que son 6 puntos de operación distintos) es más que suficiente para obtener resultados satisfactorios. Lo mismo ocurre (y con mucho más margen) para la discretización de la altura[3].

El dataset XS, como cabía esperar, presenta unos valores mayores de MSE. Ocurre algo

[2]Para la división del dataset en los conjuntos de entrenamiento y test se empleaba un método que aproximaba al entero más cercano los porcentajes de 80 y 20 %, respectivamente.

[3]La influencia de la altura en la dinámica del sistema era mucho más marginal, y lo mismo es verdad para su influencia en las ganancias del controlador.

curioso durante el entrenamiento de las redes con este dataset, y es que se llega al mayor *overfitting* encontrado en todo el estudio. Debido a que el número de puntos del dataset es tan reducido (solo 9), la división del conjunto de entrenamiento se queda con 7 puntos y son solo 2 los que se reservan para la fase de test, y estos son valores demasiado exagerados como para que la fase de test sea de utilidad. La red se aprende en exceso las ganancias correspondientes a sus 7 puntos de entrenamiento (llegando a funciones de coste durante la fase de entrenamiento del orden de la millonésima) y es el algoritmo de separación aleatoria en entrenamiento y test quien se encarga de determinar cuáles son los dos puntos que se considerarán para el test, que pueden ser más o menos distintos de los otros 7 del dataset. Si, por casualidad, los puntos de test fueran dos que pertenecieran a la columna central de velocidades (22,5 m/s, recuérdese el dataset XS), como la red habría entrenado con tres puntos a cada lado (3 a 10 m/s y otros tres a 35 m/s) además del punto restante a 22,5 m/s, sería más fácil que la «interpolación» de ganancias proporcionara un MSE de test más bajo (un mejor resultado). Si, por el contrario, los puntos de test se encontraran por azar en uno de los extremos en velocidad (por ejemplo, a bajas velocidades), la red habría aprendido sobre unos sistemas a mayor velocidad (con 6 de sus 7 puntos de entrenamiento) y no sería tan capaz de extrapolar los resultados a las menores velocidades, resultando en un mayor error MSE.

Aún así, que con un dataset de tan solo 9 puntos totales se puedan conseguir funciones de coste de centésimas es algo más que remarcable. Esto es debido a que las ganancias del controlador óptimo a lo largo de la envolvente de vuelo varían de una forma más o menos monótona, y hay algunas de ellas que experimentan variaciones tan solo del orden de ellas mismas. Esto se puede observar gráficamente representando las ganancias del controlador para distintas alturas y en función de la velocidad: la figura 5.2 lo hace para todos los puntos del dataset S. La notación que se ha empleado llama a cada ganancia (cada componente) según su posición en la matriz del controlador de la siguiente forma:

$$K = \begin{bmatrix} k_{11} & k_{12} & k_{13} & k_{14} & k_{15} \\ k_{21} & k_{22} & k_{23} & k_{24} & k_{25} \end{bmatrix}$$

Como se puede observar, hay ciertas ganancias que toman valores del orden de las varias unidades (como k_{12}, k_{13}, k_{22} y k_{23}), mientras que el resto de componentes se mantienen próximas a las décimas o incluso centésimas de unidad. Curiosamente, por otro lado, son estas ganancias menores las que más varían con la altura y la velocidad. Pero la figura sirve para comprobar el comportamiento monótono que tiene la evolución de las ganancias con la velocidad. Es por ello que un dataset reducido consigue obtener resultados satisfactorios: con pocos puntos se puede replicar con escaso error la evolución de las ganancias ya que su comportamiento no presenta irregularidades al cambiar la velocidad y, por tanto, se puede realizar una «interpolación» apropiada sin tener en complicaciones. Volviendo a la figura, se puede incluso aventurar que un punto a altas velocidades y otro a bajas sería suficiente, pues las evoluciones son más o menos rectas. Es por ello que los tres puntos en velocidad del dataset XS eran suficientes. La influencia de la altura, como

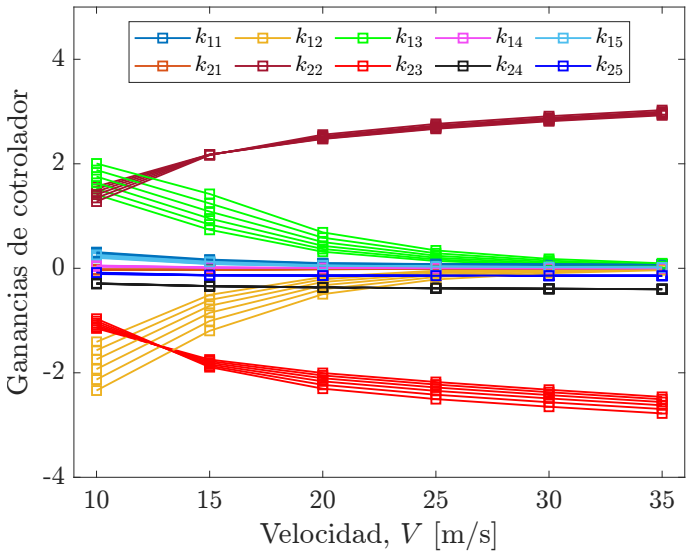


Figura 5.2: Ganancias del controlador para distintas alturas (entre 0 y 6000 m) y en función de la velocidad. Las componentes K_{ij} denotan su posición en la matriz del controlador: fila i y columna j. Cada color identifica a ganancia y las distintas alturas se corresponden a las 6 curvas de la familia de cada color.

siempre, es menor que la de la velocidad.

De la misma forma, la introducción de los parámetros másicos en la variación de estas ganancias en el problema de 5 dimensiones no deteriora el carácter monótono de dicha evolución y tampoco altera demasiado los valores concretos de las ganancias. Esto justifica que la red funcione bien también en ese caso. Esto, por otra parte, puede deberse a que el 25 % de variación estudiado para estos parámetros no logre alterar lo suficiente la dinámica del sistema como para que se produzcan variaciones significativas.

5.5. Análisis de las implicaciones de una situación real

En la sección anterior se ha demostrado que, efectivamente, una malla de la envolvente de vuelo con tan solo 36 puntos totales es suficiente para proporcionar resultados más que adecuados. En este apartado se va a realizar una corrección para presentar unos resultados má acordes a una situación real.

Si se recuerda el capítulo anterior, la sección 4.5 realizaba una comparativa entre el desempeño en el problema de seguimiento de instalar un único controlador, otro definido por intervalos de operación o la opción de implementar un control adaptativo por planificación de ganancias o mediante la red neuronal. El resultado que se quiere recuperar de dicha comparativa es que el controlador de la red neuronal conseguía un mejor comportamiento que la planificación de ganancias (además de, por supuesto, superar a los otros dos métodos).

Sin embargo, esta comparación era algo injusta desde el punto de vista de la plani-

ficación de ganancias. La red que se utilizó en aquél caso se había generado a partir del dataset A, que realizaba una discretización muy fina de la velocidad (cada 0,25 m/s) y la planificación de ganancias solo se implementó con un controlador que tenía los *breackpoints* sobre los que interpolar separados cada 5 m/s. Es de entender que realizando una planificación con una discretización más fina, como ya se indicó en dicho apartado, mejoraría la respuesta.

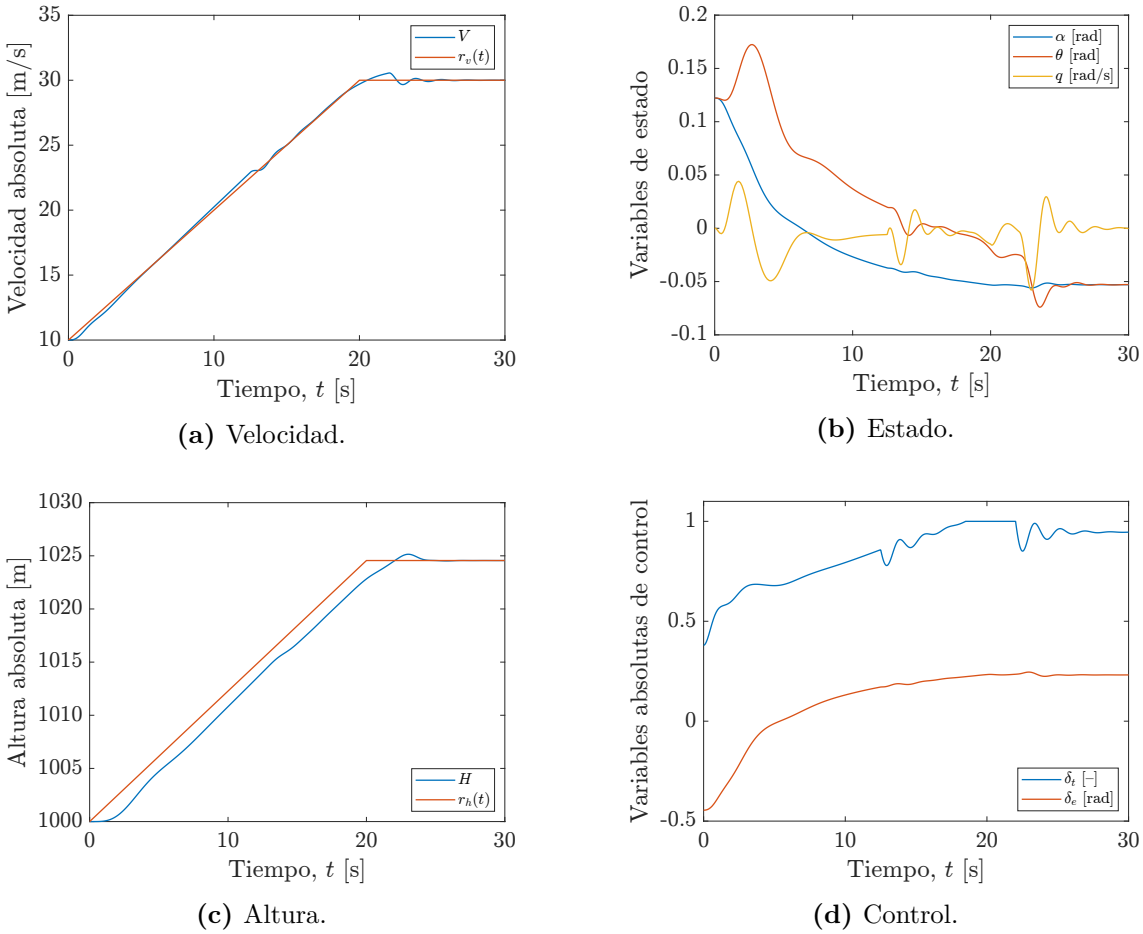
No obstante, en lugar de movernos hacia un mallado más fino, vamos a analizar la dirección contraria. En el caso de querer implementar estas técnicas para el control una aeronave real, resulta más realista pensar en tener que diseñar controladores para pocos puntos de operación en lugar de para muchos. Recuérdese que la manera de generar el dataset para entrenar a la red es diseñar un controlador en cada uno de los puntos de la malla de la envolvente y que la forma adaptada en este estudio es aplicar unas matrices de costes generales y constantes al método de LQR. Así, nuestra generación de un dataset más fino solo implica reducir el mallado y asumir un mayor tiempo de cálculo. En un caso real, el diseño del controlador en cada punto incluye muchas más consideraciones que dificultan y alargan el proceso. Estas pueden incluir descripciones más detalladas de la dinámica del avión, restricciones debidas a la resistencia estructural, límites establecidos por las Normas de Aeronavegabilidad, etc. Por ello, es razonable pensar que una situación más óptima sería considerar el menor número de puntos de diseño posible.

Teniendo esto en cuenta, se va considerar el «diseño real» de 9 controladores en toda la envolvente de vuelo (los correspondientes al dataset XS)[4]. Así, se vuelve a simular el problema de seguimiento de las dos rampas en altura y velocidad empleando esta familia de controladores: tanto la planificación de ganancias como la red neuronal tendrán que obtener las ganancias del controlador óptimo en cada instante interpolando los valores conocidos de tan solo esos 9 puntos. La figura 5.3 muestra los resultados para la planificación de ganancias y la 5.4 los de la red neuronal.

Este nuevo modelo de planificación de ganancias, en primer lugar, pese al escaso número de puntos, también logra un comportamiento más que adecuado. Comparando la respuesta de la velocidad de cabeceo de esta simulación con la que utilizaba 5 puntos de interpolación en velocidad (figura 4.14 original) se puede ver que este modelo (además de las oscilaciones habituales al principio y final de la rampa) presenta la mayor irregularidad hacia los 12 segundos, momento en que se alcanzan los 22,5 m/s, punto central de los de interpolación. En el modelo original se encontraban irregularidades con mayor frecuencia (pues había más puntos de interpolación) pero que eran de menor magnitud.

Si se compara la respuesta anterior con la de la red (figura 5.4) se vuelve a comprobar que esta consigue reducir las irregularidades comentadas y que mantiene una dinámica sin velocidad de cabeceo durante la mayor parte de las rampas de ascenso y aceleración. Por otro lado, esta respuesta con el dataset reducido es prácticamente idéntica a la del dataset original (figura 4.15).

[4]Este «diseño real» trata de ser más preciso en cuanto al número de controladores y no en cuanto a la forma de generarlos, pues el dataset XS se obtiene con el mismo LQR utilzado hasta ahora.

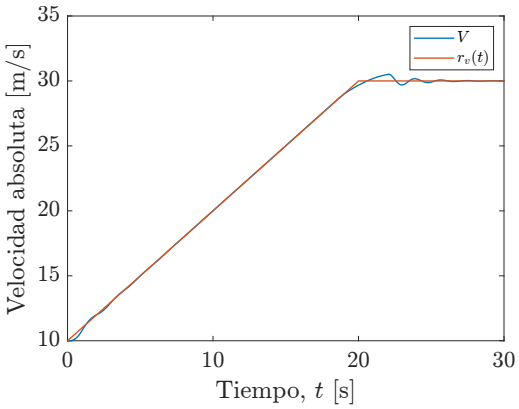


(a) Velocidad. **(b)** Estado.

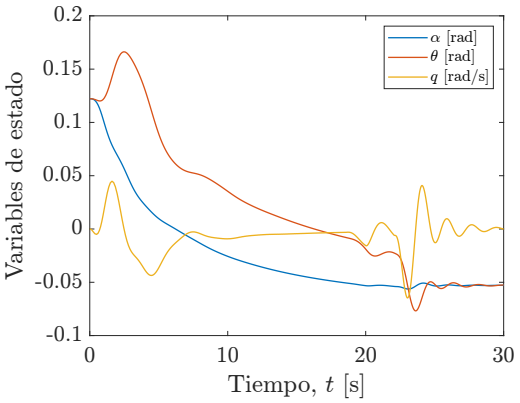
(c) Altura. **(d)** Control.

Figura 5.3: Respuesta del UAV a un sistema de tracking de la velocidad ($r_v(t)$, figura (a)) y la altura ($r_h(t)$, figura (b)) empleando planificación de ganancias a partir de la familia de controladores del dataset XS.

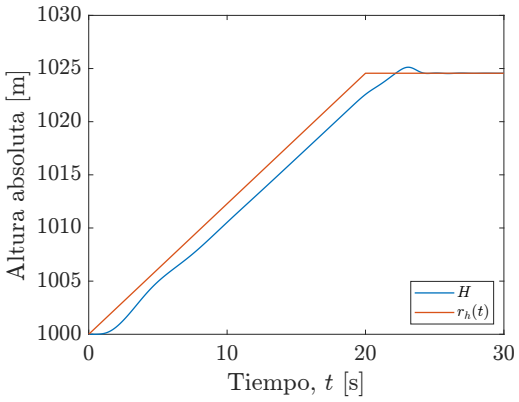
De este análisis se obtiene que ambos sistemas (planificación de ganancias y red neuronal) funcionan lo suficientemente bien en este modelo de UAV sencillo como para que sea suficiente diseñar un número reducido de puntos. Por otro lado, se ha visto que la red ha vuelto a ser la mejor solución al problema de seguimiento.



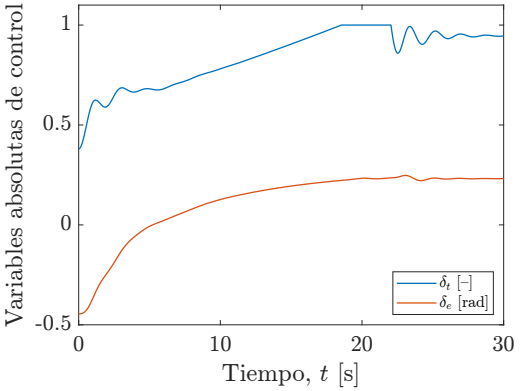
(a) Velocidad.



(b) Estado.



(c) Altura.



(d) Control.

Figura 5.4: Respuesta del UAV a un sistema de tracking de la velocidad ($r_v(t)$, figura (a)) y la altura ($r_h(t)$, figura (b)) empleando el controlador que proporciona la red neuronal (20/40/40/20) a partir del dataset XS.

Capítulo 6

Conclusiones

6.1. Resumen de los principales resultados

El objetivo de este trabajo era estudiar las redes neuronales como una posibilidad para mejorar el comportamiento de un avión en vuelo. La conclusión principal es que sí: las redes neuronales pueden desempeñar esta tarea satisfactoriamente. Se ha comprobado que alimentando a una red con la velocidad y la altura de la aeronave en cada instante se pueden obtener las ganancias del controlador que debiera llevar en avión en dichas condiciones, y las simulaciones del comportamiento del UAV de estudio tras implementar el método han revelado respuestas más que exitosas.

Para desarrollar este controlador adaptativo, en primer lugar, no es necesario que la arquitectura de la red sea muy compleja: incluso las redes con mayor número de capas que se han estudiado tienen menos de 4000 parámetros, y esto es un número realmente reducido si se compara con los tamaños que requieren las redes en otras aplicaciones. Además, incluso redes extremadamente simples con una única capa oculta de tan solo 10 neuronas han demostrado obtener resultados satisfactorios.

Por otro lado, el número de puntos de entrenamiento de la red, es decir, de controladores de referencia, también puede ser extraordinariamente pequeño. Se han obtenido resultados comparables a discretizaciones más finas de la envolvente de vuelo utilizando tan solo 6 (o incluso 3) puntos en las dimensiones de velocidad y altura.

En cuanto a estos «resultados satisfactorios» se ha observado que un error pequeño en la fase de test de la red no implica necesariamente un comportamiento igual de favorable del Índice de Desempeño (PI) del controlador frente a una perturbación. Si se quiere replicar este aspecto, se encuentra que hacen falta discretizaciones más finas o redes más complejas para compensar mallados menos densos del entorno de operación (aunque estas redes más complejas nunca superan dichos 4000 parámetros). Estas discrepancias dependerán, naturalmente, de cómo se defina dicho índice. Además, la red neuronal suaviza las diferencias de PI a lo largo de la envolvente de vuelo: el controlador que obtiene la red, de forma general, mejora las peores respuestas del sistema a bajas velocidades mientras

que penaliza en cierta medida los mayores índices que se daban a altas velocidades con el controlador de referencia. Esto quiere decir que la red está aprendiendo algo similar a «un controlador medio» de toda la envolvente de vuelo.

Siguiendo con lo anterior, se ha encontrado que, a pesar de dichas diferencias del desempeño de la red frente a una perturbación, incluso las redes más sencillas entrenadas con los datasets más reducidos consiguen un comportamiento más que notable en el problema de seguimiento. Es de remarcar que, además, la red neuronal solo se había entrenado con situaciones de vuelo horizontal rectilíneo y uniforme y que la responsabilidad del seguimiento recaía el bucle de realimentación exterior. Véase el siguiente apartado para futuras consideraciones relativas a este aspecto.

No puede olvidarse, por otro lado, la comparativa entre emplear la red en el sistema de seguimiento u optar por la planificación de ganancias. Los resultados en este campo se han inclinado del lado de la red neuronal, ya que conseguía reducir en mayor medida las oscilaciones de cabeceo a lo largo de la maniobra. Sin embargo, es de destacar también el buen desempeño del sistema de Gain Scheduling que se ha implementado en Simulink.

Otro aspecto del estudio de vital relevancia es la escalabilidad del modelo. Se ha visto que extender la funcionalidad de la red a cinco variables de entrada (tras incluir los parámetros geométricos y másicos del UAV) ha seguido proporcionando muy buenos resultados. Esto puede deberse, en parte, a la poca variación de las ganancias al extender el modelo a mayores dimensiones, por la especulación de que el 25 % de variación introducido no haya tenido demasiadas implicaciones.

Por último, se ha encontrado que la influencia de los distintos hiperparámetros de la red ha sido bastante limitada. Las consideraciones relativas a la responsabilidad de la arquitectura de la red debida a la falta de densidad del dataset ya se han comentado. Sin embargo, el resto de parámetros analizados (*train batch size*, *drop probability*, *learning rate* y función de activación) han tenido implicaciones meramente marginales sobre los resultados de la red. Estas incluían tiempos de entrenamiento y valores del error de test, pero que no tenían realmente una relevancia transferible al control del UAV en su vuelo.

6.2. Consideraciones para trabajos futuros

Derivado de los resultados anteriores, por otra parte, el presente trabajo deja abiertas varias líneas de estudio susceptibles de ser analizadas en un futuro.

En primer lugar, la forma que se utilizó para definir el Índice de Desempeño sirvió únicamente para ilustrar cómo un controlador funciona de forma diferente al cambiar de punto de operación y no representa una medida realista de evaluar cómo de buena es la respuesta del UAV debida al controlador. Se podría indagar en buscar formas alternativas de definir este índice, alterando las matrices de costes de la función de error del PI o diseñando otras técnicas que permitan identificar comportamientos que se quiere que tenga la aeronave. Como se sugería en el apartado correspondiente, esto podría incluir analizar la

derivada de las respuestas para estudiar la velocidad de cambio de las variables de estado y de control, o considerar los tiempos y porcentajes de pico. De la misma forma, este PI consideraba una manera global y promediada de evaluar la respuesta, pues todos los errores de las variables se combinaban y se consideraban de forma conjunta con la forma cuadrática que definían dichas matrices de costes. Una forma de obtener una idea más precisa del funcionamiento del controlador sería definir varios índices, que sirvieran para evaluar distintas características de la respuesta del UAV de una manera independiente. Esto permitiría ver de una forma más clara las situaciones de compromiso asociadas a cada diseño del controlador.

Relativo a este diseño del controlador está el asunto de la elección de las matrices de costes del método LQR. Recordando la forma de generar los datasets de entrenamiento, el controlador «óptimo» usado como referencia en cada punto se obtenía utilizando las mismas dos matrices de costes de estado y de control para llevar a cabo la optimización del LQR. Una forma de cambiar esto sería explorar formas alternativas de definir los controladores en las distintas regiones de la envolvente de vuelo (pues dependiendo de la altura y de la velocidad pueden interesar comportamientos diferentes). Esto, por otro lado, puede ayudar a aproximarnos más a lo que se describió como la «situación real» de tener un número más reducido de controladores de referencia. Sería interesante estudiar también la influencia de estos distintos diseños del controlador en el número de puntos necesarios para la generación de un dataset lo suficientemente detallado. Modificar la forma de obtener cada controlador de referencia puede alterar uno de los mayores aliados en todo este proceso: la monotonía de la variación de las ganancias. Podríamos encontrarnos evoluciones más complejas a lo largo de la envolvente que hicieran necesario emplear un mayor número de puntos de entrenamiento que pudieran recoger de forma adecuada el comportamiento interno de las componentes del controlador.

Del mismo modo, sería provechoso poner a prueba la «buena escalabilidad» del modelo. Como se ha comentado, la influencia de las tres nuevas dimensiones era algo superficial, por lo que se podría explorar extender el modelo con variables que tuvieran una mayor influencia en la dinámica del sistema: con restricciones a la capacidad motora, modificaciones drásticas en las eficiencias, etc. Otra forma de hacerlo sería dedicar un tiempo a estudiar las implicaciones de los tres parámetros estudiados según la Mecánica del Vuelo y aumentar los rangos de variación de dichos valores para que efectivamente se modifique la dinámica del UAV de forma más significativa.

Todo lo discutido en el apartado anterior encapsula de alguna manera la motivación de realizar simulaciones con un modelo más complejo. No obstante, todavía se puede emplear el modelo actual para estudiar múltiples otras maniobras. Existen infinidad de maniobras a realizar en el plano longitudinal para diseñar diferentes referencias de seguimiento. Además, como se adelantaba anteriormente, se podría aumentar el dataset para trimar y diseñar controladores en situaciones distintas del vuelo horizontal (como puede ser, por ejemplo, una senda de ascenso o de descenso). Con ello, se podría ver el comportamiento del servomecanismo de seguimiento cuando la red en su bucle interno también está

diseñada para mantener, por ejemplo, cierto ángulo de asiento de la velocidad.

Por otra parte, los resultados de emplear la planificación de ganancias distaban de lo que se podría haber anticipado. La evolución de las señales del UAV era adecuada (aunque parte del mérito lo tenga el bucle exterior del servomecanismo). Sin embargo, la velocidad de cabeceo presentaba ciertas irregularidades en los saltos de intervalo (en sus *breackpoints*) que, personalmente, no esperaba. Era de suponer que en su funcionamiento, realizando una interpolación continua de las ganancias de los 4 controladores más cercanos (2 en cada dimensión), la respuesta del UAV fuera mucho más suave que la encontrada. Sería necesario profundizar en la forma de implementarlo (aunque se ha seguido el procedimiento detallado en la documentación de Matlab y empleado los bloques de Simulink destinados a tal fin), por si fuera posible mejorar su comportamiento.

Recuperando el modelo de 5 dimensiones, una buena continuación del estudio sería diseñar «experimentos» en los que variasen las condiciones másicas del sistema: debido a un hipotético consumo de combustible (aunque se esté hablando de un UAV alimentado por baterías) o a una descarga súbita de mercancías, tratando de simular situaciones habituales de aviones cargueros. Estos experimentos se podrían comparar con cómo se desenvolvería un controlador convencional (sin la red neuronal) y determinar si realmente se consiguen mejoras destacables.

Por último, como es evidente, este estudio se podría extender a la dinámica completa de una aeronave, incluyendo su comportamiento lateral-direccional. Esto conllevaría ampliaciones en el abanico de maniobras a analizar, las dimensiones globales del problema (tanto en variables de estado como de control), aumentaría la complejidad del estudio y haría obligatoria una primera capa de entrada a la red que aceptase más variables de estado. Del mismo modo, se podría ampliar por el lado del control, estudiando la combinación de la red con estimadores de estado o técnicas más sofisticadas.

6.3. Nota del autor

Como hemos podido comprobar a lo largo de estas páginas y resumía en este capítulo de conclusiones, la complejidad de este estudio y las infinitas formas de ampliarlo han dejado muchos caminos abiertos para seguir investigando sobre el tema. Ha sido un trabajo muy enriquecedor en el que he desarrollado nuevas competencias, reforzado otras y explorado técnicas y conceptos interesantes, obteniendo una primera aproximación, en mi opinión, rica y detallada.

Es por ello que, motivado por el interés que ha desertado en mí este TFG y la belleza de las disciplinas involucradas, recomiendo encarecidamente a cualquier lector interesado coger las riendas del estudio en la dirección que encuentre más atractiva. Hay recomendaciones de estudio más asequibles que otras, claro está, pero el simple hecho de tratar de replicar algunos de los resultados que aquí presento puede ser de mucho interés.

Para ello, además de los códigos más relevantes que se recogen en los siguientes apéndices, pongo a disposición del público interesado todos los códigos, archivos, modelos y figuras utilizados para el estudio y la generación del contenido gráfico de este documento. Está disponible en el repositorio de GitHub creado a tal efecto en el link:

`https://github.com/IgnVidArc/TFG-repo`

Por otro lado, será un placer recibir cualquier consideración adicional, recomendación, crítica o duda en mi correo personal:

`ignacio.vidal.arce@gmail.com`

Esto ha sido todo.

Ignacio Vidal de Arce
Madrid, junio de 2023.

Apéndice A

Códigos de Matlab

Este primer apéndice está dedicado a recoger varios de los códigos de Matlab que puedan ser útiles para el lector interesado. El siguiente apéndice B hace lo propio con otros códigos de Python y el C incluye los modelos de Simulink más significativos.

Estos tres apéndices están muy conectados entre sí y será frecuente que códigos de Matlab hagan referencia a códigos de Python y a modelos de Simulink. Se ha intentado aclarar el contexto de donde existe cada código o modelo relacionándolo con el resto de piezas del conjunto en la breve introducción de cada apartado.

A.1. Modelo no lineal de UAV

La función recogida en esta sección es la introducida en el apartado 2.5 del capítulo sobre el modelo de UAV.

La función `UAVNL` recibe los vectores vector de estado, `XV`, y de control, `UV`, en el instante actual y devuelve el vector de la derivada del vector de estado, `DX`. A su vez, la función acepta también el argumento `param`, que incluye la masa del UAV, su posición del centro de masas y su momento de inercia de cabeceo.

El código comienza definiendo todas las magnitudes necesarias (recogidas en las tablas de la sección 2.4) y las derivadas de estabilidad y de control y asigna las variables apropiadas a las que se introducen en el vector de parámetros modificables (`param`). Luego plantea las ecuaciones de la dinámica del sistema y termina calculando las derivadas de las variables de estado.

Un ejemplo de cómo se encontrará esta función dentro de un modelo de Simulink puede verse en la figura C.1.

Listing A.1: `UAVNL.m`: Función del modelo no lineal del UAV.

```
%% UAVNL.m
function XD = UAVNL(XV,UV,param)
% Empty output vector:
```

```
XD = zeros(5,1);

% UAV PARAMETERS
S = 0.4121;                % Wing surface [m^2]
C = 0.2192;                % Mean aerodynamic chord [m]
XCGR = 0.33;               % Reference center of gravity position [-]
g = 9.8;                   % Gravitation acceleration [m/s^2]

% Stability and control derivatives:
CL0 = 0.405;
CLalpha = 5.379292;
CLq = 11.133615;
CD0 = 0.0277;
CDalpha = 0.14897;
CDalpha2 = 0.9848;
Cm0 = 0.0418;
Cmalpha = -6.0;
Cmalphadot = -7.0;
Cmde = -1.552;
Cmq = -18.422237;

% State variables:
VT    = XV(1);          % Speed [m/s]
ALPHA = XV(2);          % Angle of atack [rad]
THETA = XV(3);          % Pitch angle [rad]
Q     = XV(4);          % Pitch angular velocity [rad/s]
H     = XV(5);          % Height [m]

% Control variables:
Dt = UV(1);             % Thrust position [-]
De = UV(2);             % Elevator position [rad]

% Aditional parameters:
M   = param(1);
XCG = param(2);
IP  = param(3);

% Modified elevator control coefficient:
deltaXCG = XCG - XCGR;
Cmde = Cmde * (.80 - deltaXCG*C)/.80;

% Density at altitude H:
d = density(H);

% Previous calculations
U = VT*cos(ALPHA);      % X-body velocity component
W = VT*sin(ALPHA);      % Z-body velocity component
PDS = 0.5*d*(VT^2)*S;   % Dynamic pressure * Wind surface [Pa]

% Proupulsion parameters:
Dh = 0.254;             % Propeller diameter [m]
REVmax = 222;           % Max revolutions [rev/s]
```

```
CT0 = 0.13805;          % Traction coefficients
CTj = -0.2049;
REV = REVmax*Dt;        % Instantaneous revolutions [rev/s]
J = VT/(REV*Dh);        % J coeffient [-]
C_T = CT0 + CTj*J;      % Traction coefficient [-]
T = C_T*d*REV^2*Dh^4;   % Traction [N]

% Forces and moment coefficients:
CLT = CL0 + CLalpha*ALPHA + CLq*Q*C*0.5/VT;
CDT = CD0 + CDalpha*ALPHA + CDalpha2*ALPHA^2;
CZ = -CDT*sin(ALPHA) - CLT*cos(ALPHA);
CmT = Cm0 + (Cmalpha*ALPHA) + (Cmde*De) + (CZ*(XCG-XCGR)) ...
+ (C*0.5*Cmq*Q/VT);

% Aerodynamic forces:
L = CLT*PDS;
D = CDT*PDS;

% System of differential equations:
UDOT = -Q*W + (T - M*g*sin(THETA) - D*cos(ALPHA) + L*sin(ALPHA))/M;
WDOT = Q*U + (M*g*cos(THETA) - D*sin(ALPHA) - L*cos(ALPHA))/M;
ALPHADOT = (U*WDOT - W*UDOT)/VT^2;
CmT = CmT + Cmalphadot*C/(2*VT)*ALPHADOT;
QDOT = CmT*PDS*C/IP;
VTDOT = (U*UDOT + W*WDOT)/VT;

% Output vector:
XD(1) = VTDOT;
XD(2) = ALPHADOT;
XD(3) = Q;
XD(4) = QDOT;
XD(5) = VT*sin(THETA-ALPHA);
```

A.2. Trimado, linealización y diseño del controlador

Este código hace referencia a lo discutido en la sección 3.3 y trata sobre el trimado del sistema en unas condiciones determinadas y la linealización del sistema alrededor de dicha posición de equilibrio. Las últimas líneas también obtienen el controlador por LQR definitivo de la sección 3.5.

Se comienza definiendo los parámetros necesarios de la configuración másica del UAV y su condición de vuelo. Después se crean los vectores necesarios para la selección de las componentes necesarias para la función `trim`. Se obtienen las matrices del sistema gracias a `linmod` y se acaba generando las funciones de coste del controlador para su posterior cálculo.

El sistema `UAVTrimh` al que hacen referencia las funciones del código es un modelo de Simulink que contiene la función no lineal del UAV y que se muestra en la figura C.1.

Listing A.2: `trimlinlqr.m`: Trimado, linealización y diseño del controlador.

```
%% trimlinlqr.m
% Parameters:
W = 2.5;     % Mass [kg]
XCG = 0.33; % Position of center of gravity [-]
IP = 0.1568;% Moment of inertia [kg m^2]
% Trim conditions:
V = 15;      % Velocity [m/s]
H = 1000;    % Altitude [m]
% Trim:
X0 = [V;0;0;0;H];
U0 = [0.5;-0.1];
Y0 = [];
IX = [1,4,5];
IU = [];
IY = [];
[Xtrim,Utrim,Ytrim,DXtrim] = trim('UAVTrimh',X0,U0,Y0,IX,IU,IY);
% Linear Model:
[A,B,C,D] = linmod('UAVTrimh',Xtrim,Utrim);
% Cost matrices:
Q = blkdiag(1,100,100,100,10);
R = blkdiag(100,500);
% Controller:
K = lqr(A,B,Q,R);
```

A.3. Sistema de tracking

Este código del siguiente listing lleva a cabo el proceso de generación de las matrices del bucle interno y del error del sistema de seguimiento del apartado 3.6. Se comienza, como con el listing anterior, obteniendo el punto de equilibrio y las matrices del sistema linealizado. Se definen las matrices de costes de ambos casos de seguimiento (solo la velocidad y la pareja velocidad y altura) y se calculan las matrices `K1` y `K2` necesarias para el sistema.

Para ver la aplicación concreta y el funcionamiento de estas matrices, véase el modelo de la sección C.4.

Listing A.3: `tracking.m`: Diseño del sistema de tracking.

```
%% tracking.m
% Parameters:
W = 2.5;     % Mass [kg]
XCG = 0.33; % Position of center of gravity [-]
IP = 0.1568;% Moment of inertia [kg m^2]
% Trim conditions:
V = 15;      % Velocity [m/s]
H = 1000;    % Altitude [m]
% Trim:
X0 = [V;0;0;0;H];
U0 = [0.5;-0.1];
Y0 = [];
IX = [1,4,5];
IU = [];
IY = [];
[Xtrim,Utrim,Ytrim,DXtrim] = trim('UAVTrimh',X0,U0,Y0,IX,IU,IY);
% Linear Model:
[A,B,C,D] = linmod('UAVTrimh',Xtrim,Utrim);

% ONLY FOLLOWING THE VELOCITY
% Selection of the followed variable:
Ct = [1, 0, 0, 0, 0];
% Augmented matrices:
At = [A, zeros(5,1); -Ct, 0];
Bt = [B; 0 0];
% Cost matrices:
Qt = blkdiag(1, 100, 100, 100, 10, 100); % Augmented state
Rt = blkdiag(100,500);
Kt = lqr(At,Bt,Qt,Rt);

K1 = Kt(:,1:5);
K2 = Kt(:,6);

% FOLLOWING BOTH THE VELOCITY AND ALTITUDE
% Selection of the followed variables:
Ct = [1, 0, 0, 0, 0;
0, 0, 0, 0, 1];
```

```
% Augmented matrices:
At = [A, zeros(5,2); -Ct, zeros(2,2)];
Bt = [B; 0 0; 0 0];
% Cost matrices:
Qt = blkdiag(1, 1000, 1000, 100, 10, 100, 5); % Augmented state
Rt = blkdiag(100,100);
Kt = lqr(At,Bt,Qt,Rt);

K1 = Kt(:,1:5);
K2 = Kt(:,6:7);
```

A.4. Generación del dataset

A continuación se recoge el proceso definido en el apartado 4.4 sobre la generación del dataset. El resultado del código presentado es un fichero `dataset.csv` donde las dos primeras columnas identifican velocidad y altura, y las 10 siguientes recogen las 10 ganancias (ordenadas por filas de K), de la matriz del controlador.

Listing A.4: `GenerateDataset.m`: Generación del dataset.

```
%% GenerateDataset.m
% Additional parameters:
W = 2.5;     % Mass [kg]
XCG = 0.33; % Position of center of gravity [-]
IP = 0.1568;% Moment of inertia [kg m^2]
% LQR weight matrices:
Q = blkdiag(1, 1000, 1000, 100, 10);
R = blkdiag(100,500);
% General settings for trim:
U0=[1;0.5]; Y0=[]; IX=[1 4 5]; IU=[]; IY=[];
% FLIGHT ENVELOPE MESH:
vV = 10:0.25:30;
vH = 100:100:3100;
% Controller weights:
CONTROLLER = zeros(length(vV), length(vH), 2, 5);
% Loop:
for i = 1:length(vV)
    for j = 1:length(vH)
        % Conditions
        V = vV(i); H = vH(j);
        % Trim:
        X0=[V;0;0;0;H]; U0=[0.5;-0.1];
        [Xtrim,Utrim,Ytrim,DXtrim] = trim('UAVTrimh',X0,U0,Y0,IX,IU,IY);
        % Linear Model:
        [A,B,C,D]=linmod('UAVTrimh',Xtrim,Utrim);
        % Controller:
        [K,S,P] = lqr(A,B,Q,R);
        CONTROLLER(i,j,:,:) = K;
    end
end
% Dataset:
name_dataset = '../Datasets/dataset.csv';
export = zeros(length(vV)*length(vH), 2 + 2*5);
idx = 0;
for i = 1:length(vV)
    for j = 1:length(vH)
        idx = idx + 1;
        export(idx,:) = [vV(i), vH(j), ...
            reshape(CONTROLLER(i,j,1,:),1,5), ...
            reshape(CONTROLLER(i,j,2,:),1,5)];
    end
end
```

```
writematrix(export,name_dataset,'Delimiter',',')
```

A.5. Funciones del los subsistemas del controlador discreto y de la red neuronal

Esta sección incluye, en primer lugar (listing A.5), la función que permite obtener el controlador discreto a emplear para un estado de vuelo que aparece en el subsistema de la figura C.6. La función comprueba en qué rango de velocidades del argumento `Vs_range` se encuentra la velocidad que lleva el UAV en ese momento y escoge el controlador asociado al actual intervalo.

Por otro lado, el listing A.6 es el pequeño código que usa el modelo de tracking del apartado C.5 para utilizar la red neuronal. Este código de Matlab se comunica con Python llamando al archivo `state2k.py` (listing B.2), que es el que contiene realmente el funcionamiento de la red neuronal.

Listing A.5: `state2k_discrete.m`: Función del controlador discreto.

```
%% state2k_discrete.m
function K = state2k_discrete(state, Ks_range, Vs_range)
    % state = [V; H]
    % Ks_range = [K1, K2, K3, ...], with dimensions (2,5,[])
    % Vs_range = [[V_start1;V_end1], [V_start2;V_end2],...]
    % Range of V in wich the controller to use is K3 is [Vs_range(1,3),
       Vs_range(2,3)]
    % K2 is used in [V_start2, V_end2).
    % Example: to use a controller K1 betwwen 10 and 15 m/s, another K2
    % between 15 and 20, K3 between 20 and 25, and K4 between 25 and
    % 30 m/s, the arguments would be:
    % Ks_range = [K1, K2, K3, K4]
    % Vs_range = [[10;15], [15;20], [20;25], [25;30]];
    % NOTE: Function created only for speed discretizations

    V = state(1);
    H = state(2);

    K = zeros(2,5);

    Ks = reshape(Ks_range,2,5,[]);
    found = 0;

    for i = 1:size(Vs_range,2)
        range = Vs_range(:,i);
        if V >= range(1) && V < range(2)
            K = Ks(:,:,i);
            found = 1;
            disp(i)
        end
    end

    if ~found
```

```
        K = zeros(2,5);
        disp('ERROR: state2k_discrete: Range not satisfied.')
    end
end
```

Listing A.6: `state2k.m`: Función del controlador con la red neuronal.

```
%% state2k.m
function K = state2k(state)
    % Calling the python script to obtain the controller (K)
    % given the current state (state).

    pyfile = '../Networks/state2k.py';
    model = pyrunfile(pyfile, 'model');
    K = double(pyrunfile(pyfile, 'K', model=model, state=state));
end
```

Apéndice B

Códigos de Python

B.1. Creación, entrenamiento y evaluación de la red neuronal

Aquí se presenta el código de Python empleado para la lectura y tratamiento del dataset generado en el listing A.4, la definición de la red según la sección 4.4, el entrenamiento y evaluación de la misma junto con la exportación del modelo para su incorporación en Matlab y Simulink.

Se emplea la librería de Machine Learning `ScikitLearn`[1] para el preproceso de las variables de entrada de la red (principalmente para su normalización). Por encima de la arquitectura de `PyTorch`[2] para definir la arquitectura de la red, se usa `PyTorch Lightning`[3] para automatizar el proceso de entrenamiento y de evaluación. Después, se guardan varios diccionarios con la estructura y con la información de los parámetros de cada una de las capas para su posterior uso durante la simulación en Simulink.

Listing B.1: `NNScript.py`: Entrenamiento de la red neuronal.

```
## NNScript.py
# PACKAGES

# data handling
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error

# deep learning
import torch
import torch.nn as nn
import torch.nn.functional as F
```

[3]Documentación de `ScikitLearn` en: https://scikit-learn.org/stable/
[3]Documentación de `Pytorch` en: https://pytorch.org/docs/stable/index.html
[3]Documentación de `Pytorch Lightning` en: https://lightning.ai/docs/pytorch/stable/

```
from torch.utils.data import Dataset, DataLoader, random_split
import pytorch_lightning as pl
from pytorch_lightning.callbacks.early_stopping import EarlyStopping

# utilities:
import copy
import matplotlib.pyplot as plt
import pandas as pd
from joblib import dump


# %% DATASET CLASSES AND METHODS
class GetDataset(Dataset):
    "Geting features and outputs ndarray from file."
    def __init__(self, filename, ninputs, noutputs, delimiter: str=','):
        # data loading
        xy = np.loadtxt(filename, delimiter=delimiter, dtype=np.float32)
        self.X = xy[:, :ninputs]
        self.y = xy[:, ninputs:ninputs+noutputs]
        self.n_samples = xy.shape[0]

    def __len__(self):
        return self.n_samples

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]


class DatasetFromXy(Dataset):
    def __init__(self, X, y):
        self.X = torch.from_numpy(X)
        self.y = torch.from_numpy(y)
        self.n_samples = X.shape[0]

    def __len__(self):
        return self.n_samples

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]


# %% DATASET GENERATION
filename = '../Datasets/dataset_005.csv'
ninputs = 2
noutputs = 10

data_raw = GetDataset(filename=filename, ninputs=ninputs, noutputs=noutputs
    )
X_raw = data_raw.X
y_raw = data_raw.y

scalerX = StandardScaler()
```

```
X_scaled = scalerX.fit_transform(X_raw)

data_scaled = DatasetFromXy(X_scaled, y_raw)

test_set, train_set = random_split(data_scaled, lengths=[.2,.8], generator=
    torch.Generator())

print(f'Number of training samples: {len(train_set):4}\nNumber of testing
    samples: {len(test_set):5}')
nsamples = len(test_set)

# %% DATALOADERS AND TRAIN/TEST BATCHES
train_batch_size = 100
test_batch_size = 20

train_loader = DataLoader(train_set, batch_size=train_batch_size, shuffle=
    True, num_workers=0)
test_loader = DataLoader(test_set, batch_size=test_batch_size, shuffle=
    False, num_workers=0)


# %% NEURAL NETWORK DEFINITION
class Network(pl.LightningModule):
    def __init__(self, input_size, output_size, hidden_layers,
                 learning_rate, drop_p=0.2, leaky_slope=0.0):
        """ Builds a fully connected network with arbitrary hidden layers.

            Arguments
            ---------
            input_size: integer, size of the input layer.
            output_size: integer, size of the output layer.
            hidden_layers: list of integers, the sizes of the hidden layers
                .
            learning_rate: learning rate for the optimizer.
            drop_p: float = 0.2, drop out probability.
            leacky_slope: float = 0.0 slope of the leakyReLU activation
                function.
            A value of zero is equivalent to using the ReLU function.
        """
        super().__init__()

        self.input_size = input_size
        self.output_size = output_size
        self.hidden_sizes = hidden_layers
        self.structure_dict = {'input_size': input_size,
                               'output_size': output_size,
                               'hidden_sizes': hidden_layers}

        # Input to a hidden layer
        self.hidden_layers = nn.ModuleList([nn.Linear(input_size,
            hidden_layers[0])])
```

```
        # Add a variable number of more hidden layers
        layer_sizes = zip(hidden_layers[:-1], hidden_layers[1:])
        self.hidden_layers.extend([nn.Linear(h1, h2) for h1, h2 in
            layer_sizes])

        # Ouput layer
        self.output = nn.Linear(hidden_layers[-1], output_size)

        # Dropout probabilty
        self.dropout = nn.Dropout(p=drop_p)

        # Loss function
        self.loss_fun = nn.MSELoss()    # Mean Squared Error Loss

        # Optimizer learning rate
        self.learning_rate = learning_rate

        # optimizer:
        self.optimizer = []



    def forward(self, x):
        """ Forward pass through the network, returns the output logits."""
        for each in self.hidden_layers:
            x = F.relu(each(x))
            x = self.dropout(x)
        x = self.output(x)
        return x


    def configure_optimizers(self):
        optimizer = torch.optim.Adam(self.parameters(), lr=self.
            learning_rate)
        self.optimizer = optimizer
        return optimizer


    def training_step(self, train_batch, batch_idx):
        "Definition of the training loop."
        X, y = train_batch          # (extracting features and outputs)
        # y = y.type(torch.float32) # (just in case)
        # forward pass
        y_pred = self.forward(X).squeeze()
        # compute loss
        loss = self.loss_fun(y_pred, y)
        self.log_dict({'train_loss': loss}, on_step=False, on_epoch=True,
            prog_bar=True,
                      logger=True, enable_graph=True)


    def test_step(self, test_batch, batch_idx):
```

```
        X, y = test_batch
        # forward pass
        y_pred = self.forward(X).squeeze()
        # compute metrics
        loss = self.loss_fun(y_pred, y)
        r2 = r2_score(y_pred, y)
        rmse = np.sqrt(mean_squared_error(y_pred, y))
        self.log_dict({'Cost function': loss, 'r2': r2, 'Root Mean Square
            Error': rmse},
                      on_step=False, on_epoch=True, prog_bar=True, logger=
                          True)
        return loss

# %% NETWORK CREATION

# Number of layers and their sizes:
input_size = ninputs
output_size = noutputs
hidden_layers = [20, 40, 40, 20]

# Other hyperparameters:
max_epochs = 500
lr = 0.001
drop_p = 0.1
leaky_slope = 0.0

model = Network(input_size=input_size, output_size=output_size,
    hidden_layers=hidden_layers,
                learning_rate=lr, drop_p=drop_p)

# Including early stoping: 'patience' is the key parameter here.
patience = 20
early_stop_callback = EarlyStopping(monitor="train_loss", min_delta=0.00,
    patience=patience, verbose=True, mode="min")

# %% DEFINITION OF THE TRAINER
trainer = pl.Trainer(accelerator='cpu', devices='auto', max_epochs=
    max_epochs,
                     callbacks=[early_stop_callback], log_every_n_steps=8)

# TRAINING
trainer.fit(model=model, train_dataloaders=train_loader)
# writer.flush()
# writer.close()


# %% TESTING
trainer.test(model=model, dataloaders=test_loader)

# %% Saving & Loading Model to Disk
# ----- IDENTIFIER ------
identifier = '005_20_40_40_20'
```

```
# ------ -------- -------

# SAVING THE STATE DICT
state_dict_path = '../Models/' + 'state_dict_' + identifier
torch.save(model.state_dict(), state_dict_path)

# SAVING THE STRUCTURE DICT
structure_dict = model.structure_dict
dump(structure_dict, '../Models/' + 'structure_' + identifier, compress=
    True)

# SAVING THE STANDARD SCALER
dump(scalerX, '../Models/scalerX_' + identifier[0:3] + '.bin', compress=
    True)
```

B.2. Función para la inferencia del controlador con la red neuronal desde Matlab

En esta sección se recoge el código de Python que hace posible la obtención del controlador para una condición de vuelo a partir de un modelo de red previamente entrenado y guardado en disco (obtenido con el código del apartado anterior). El código que se recoge en el listing a continuación se llama desde la función `state2k.m` de Matlab del listing A.6 y es lo que se esconde detrás del empleo de la red en el problema de seguimiento de la figura 4.15 y la tercera de las opciones de controlador del modelo C.5.

El código comienza importando los módulos necesarios. Entre ellos están los encargados de la definición de la red (la librería de Pytorch: `Torch`), el `StandardScaler` de `sklearn` para poder normalizar el estado de la misma forma que se hizo durante el entrenamiento, y la función `load` de `joblib` para poder importar de nuevo los diccionarios que se generaron tras el entrenamiento de la red al guardar la arquitectura y el valor de los parámetros de cada capa.

Se emplea el mismo identificador `identifier` que se usó al guardar la red en el código del apartado anterior. Se definen las funciones necesarias y se añaden dos sentencias al final del código: una para le creación del modelo (`model = ...`) y otra para la obtención del controlador (`K = ...`). Estas son a las que se llama desde Matlab para el uso de la red desde Simulink (vuélvase al listing A.6).

La función acepta tanto el caso de la red con dos entradas como cuando se amplía el modelo a cinco entradas incluyendo las propiedades másicas (capítulo 5). Esto se controla a partir del identificador ya mencionado: incluyendo los caracteres '5D' al inicio del identificador se elige qué red se quiere utilizar en cada caso.

Listing B.2: `state2k.py`: Función para inferir el controlador con la red neuronal.

```
## state2k.py

# PACKAGES
import sys
sys.path.append(r'C:\Users\ignac\OneDrive - Universidad Politécnica de
    Madrid\04 - GIA 4 - Cuatri 8\TFG\TFG-AI-repo\Networks')
sys.path.append(r'C:\Users\ignac\OneDrive - Universidad Politécnica de
    Madrid\04 - GIA 4 - Cuatri 8\TFG\TFG-AI-repo\Models')

# data handling
import numpy as np
from sklearn.preprocessing import StandardScaler

# deep learning
import torch
import torch.nn as nn
import torch.nn.functional as F
```

```
# to load the Standard Scaler
from joblib import load

# models path (for state_dict, structure, and scalerX) and general strings:
models_path = '../Models/'
root_name_given_to_state_dict = 'state_dict_'
root_name_given_to_structure = 'structure_'
root_name_given_to_scalerX = 'scalerX_'

# ----- IDENTIFIER ------
identifier = '005_20_40_40_20'
# ------ -------- -------

# default structure:
structure = load(models_path + root_name_given_to_structure + identifier)
# default scalerX:
scalerX = load(models_path + root_name_given_to_scalerX +
identifier[0:3] + '.bin')

#########################################################################
#########################################################################

"""Only the forward method is included in this reduced network."""
class NetworkReduced(nn.Module):
    def __init__(self, input_size, output_size, hidden_layers):
    """ Builds a fully connected network with arbitrary hidden layers.

    Arguments
    ---------
    input_size: integer, size of the input layer.
    output_size: integer, size of the output layer.
    hidden_layers: list of integers, the sizes of the hidden layers."""

    super(NetworkReduced, self).__init__()
    # Input to a hidden layer
    self.hidden_layers = nn.ModuleList([nn.Linear(input_size,
    hidden_layers[0])])
    # Add a variable number of more hidden layers
    layer_sizes = zip(hidden_layers[:-1], hidden_layers[1:])
    self.hidden_layers.extend([nn.Linear(h1, h2) for h1, h2 in
    layer_sizes])
    # Ouput layer
    self.output = nn.Linear(hidden_layers[-1], output_size)

    def forward(self, x):
    """ Forward pass through the network, returns the output logits."""
    for each in self.hidden_layers:
    x = F.relu(each(x))
    # x = self.dropout(x)
    x = self.output(x)
    return x
```

```
#########################################################################
#########################################################################

def PredictController(model, state, scalerX=scalerX):
    """
    Function that receives the trim state vector and generates a prediction
    of the appropriate controller based on the trained model 'model'.

    Arguments
    ---------
    model: Network to use
    state: State in the flight envelope to infere from
    scalerX: scaler used for the features list from the dataset
    """
    #
    state = np.array(state).reshape(1,-1)
    features = scalerX.transform(state)
    out_pred = model(torch.Tensor(features))    # torch.Tensor([1,10])
    K_comps = out_pred.detach().numpy()
    state_dim = K_comps.size    # 10
    return K_comps.reshape(2, int(state_dim/2))

#########################################################################
#########################################################################

def model_birth(identifier=identifier, structure=structure):
    """Creates a network from a saved model.state_dict with the identifier
    `identifier` in `..\Models\` with the structure stored in `structure`.
        """

    # network structure parameters:
    input_size = structure['input_size']
    output_size = structure['output_size']
    hidden_layers = structure['hidden_sizes']

    # network initialization:
    model = NetworkReduced(input_size=input_size, output_size=output_size,
        hidden_layers=hidden_layers)

    # network loading
    path = models_path + root_name_given_to_state_dict + identifier
    model.load_state_dict(torch.load(path))

    # setting the network in evaluation mode
    # (setting dropout and batch normalization layers to evaluation mode)
    model.eval()

    return model

#########################################################################
#########################################################################
```

```
def PredictController(model, state, scalerX=scalerX):
    """
    Function that receives the trim state vector and generatesa prediction
    of the appropriate controller based on the trained model 'model'.

    Arguments
    ---------
    model: Network to use
    state: State in the flight envelope to infere from
    scalerX: scaler used for the features list from the dataset
    """
    #
    state = np.array(state).reshape(1,-1)
    features = scalerX.transform(state)
    out_pred = model(torch.Tensor(features))    # torch.Tensor([1,10])
    K_comps = out_pred.detach().numpy()
    state_dim = K_comps.size    # 10
    return K_comps.reshape(2, int(state_dim/2))

########################################################################
########################################################################

# To call 'model' from Matlab:
model = model_birth(identifier=identifier)

# To call 'K' from Matlab:
if identifier[0:2] != '5D':
    state_ref = [20., 2000.]
else:
    state_ref = [20., 2000., 2.5, 0.33, 0.1568]

K = PredictController(model, state=state_ref)
```

Apéndice C

Modelos de Simulink

C.1. Modelo para trimar el UAV no lineal

En la siguiente figura se muestra el modelo de Simulink empleado para el trimado del sistema. Pueden observarse el modelo no lineal (listing A.1), un integrador, una entrada y salida y las variables introducidas como el vector de parámetros.

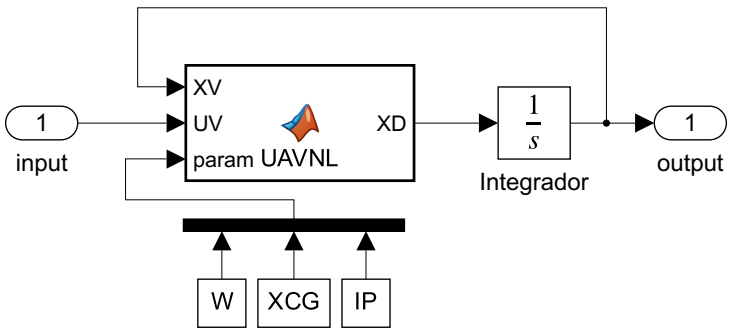


Figura C.1: `UAVTrimh.slx`: Modelo no lineal para el trimado y linealización.

C.2. Modelo de UAV lineal en bucle abierto

Modelo lineal en bucle abierto correspondiente a la discusión del apartado 3.4. En la figura C.2 se muestra la referencia del escalón en el timón de profundidad (para obtener la figura 3.3): a eliminarse en caso de querer simular la respuesta a unas condiciones iniciales (figuras 3.1 y 3.2).

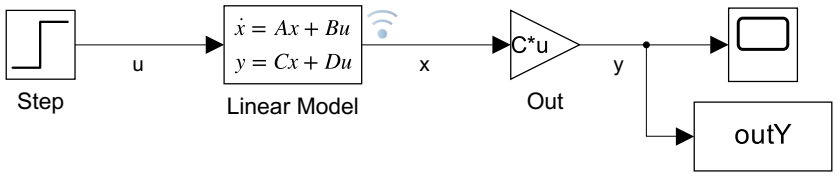


Figura C.2: `UAVLinOL_step.slx`: Modelo lineal en bucle abierto con entrada escalón.

C.3. Modelo de UAV lineal en bucle cerrado

En esta sección se muestra el principal modelo lineal empleado para las simulaciones en un único punto. Concretamente, este modelo es el empleado siempre que se trata de obtener el Índice de Desempeño de cualquier controlador (por tanto, va emparejado con el bloque `sim` de los diagramas de las figuras 4.5 y 4.3). Además, claro está, es el usado para simular cualquier respuesta a una perturbación (capítulos 3 y 4). En la figura se pueden observar el bloque del modelo lineal, el bucle de realimentación con el controlador, una referencia nula, varios bloques «Scope» (utilizados en distintos momentos de la vida de este modelo) y un bloque de saturación. Este último asegura que no se superan los límites físicos del funcionamiento de las variables de control: que, por mucho que a partir de un estado perturbado las ganancias del controlador pretendan poner el acelerador al 120 %, el límite del motor es el 100 %. Lo mismo ocurre para la deflexión geométrica del timón de profundidad.

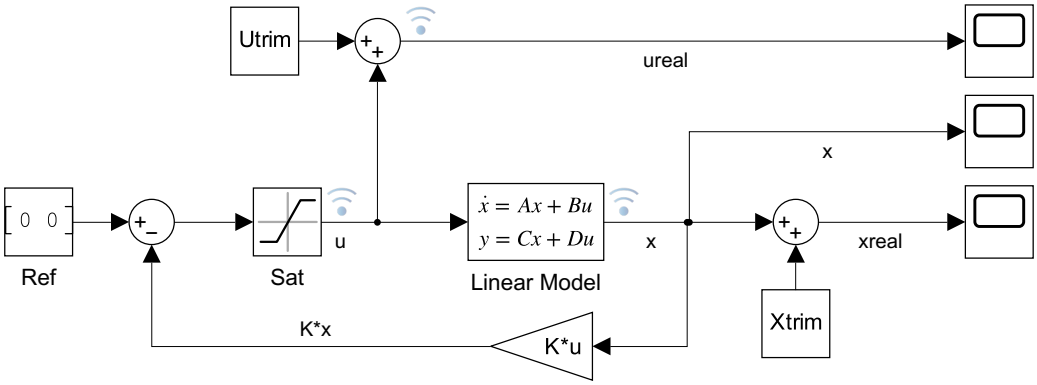


Figura C.3: `UAVLinCL.slx`: Modelo lineal en bucle abierto con entrada escalón.

C.4. Modelo de un sistema de tracking con el UAV no lineal

Este modelo hace referencia a lo discutido en la sección 3.6 sobre el sistema de seguimiento y debe ir asociado al código mostrado en el listing A.3 (para el código del sistema de tracking) y en el listing A.1 (UAV no lineal).

Se muestran, pues, todos los elementos del sistema de tracking. Se usa el modelo no lineal de UAV (debido a la variación de las condiciones de vuelo fuera del rango lineal alrededor de la situación de trimado), sus respectivos parámetros y el integrador correspondiente. Al comienzo de la señal se encuentra la referencia a seguir, cuyo error es integrado y multiplicado por la matriz de ganancias del error K_2. En el bucle superior se tiene la matriz K_1 y en la parte inferior del modelo se introduce la matriz `Ct` (C en la teoría descrita en la sección 3.6) para obtener las variables de seguimiento.

Se ha añadido el mismo saturador del modelo anterior para limitar el valor que toman las variables de control. Por otro lado, las constantes `Xtrim` y `Utrim` son necesarias debido a la combinación del modelo no lineal (que usa variables absolutas) con las realimentaciones de control (que debe multiplicar a las variables incrementales).

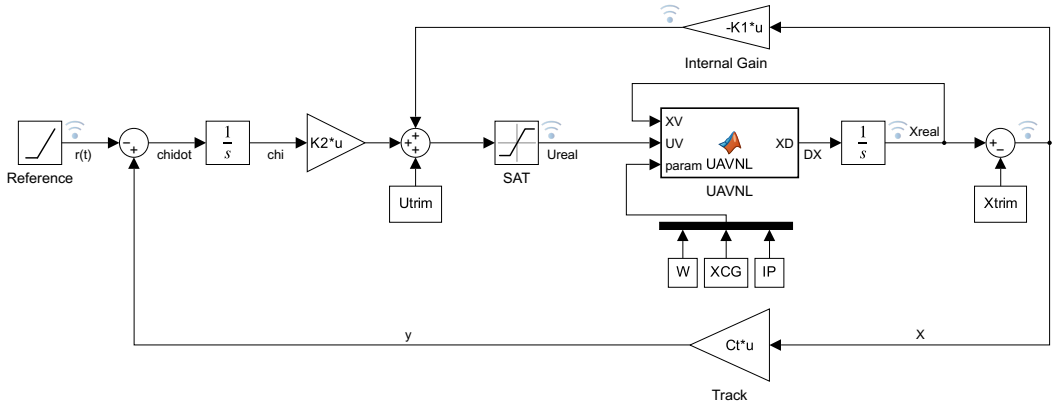


Figura C.4: `UAVNLCL_TRACK.slx`: Modelo para el sistema de tracking.

C.5. Modelo con la combinación de todos los sistemas de tracking para el UAV no lineal: controlador discreto, gain scheduling y red neuronal

En esta sección se presenta el modelo que está detrás de la obtención de todas las figuras del apartado 4.5 sobre la comparación de la red neuronal con los demás mecanismos de selección de los controladores para el sistema de seguimiento. Buena parte de los bloques mostrados son los mismos que los del sistema de tracking del apartado anterior; es en la realimentación del bucle interior donde difiere con el de la figura C.4 y donde se esconde todo el cuerpo de este modelo.

Para dicha realimentación se pueden emplear tres alternativas: el *controlador discreto*, la planificación de ganancias (*gain scheduling*) y la red neuronal. La figura C.5 muestra el modelo completo y es con los selectores de la parte superior con los que se elige qué técnica emplear para la elección del controlador.

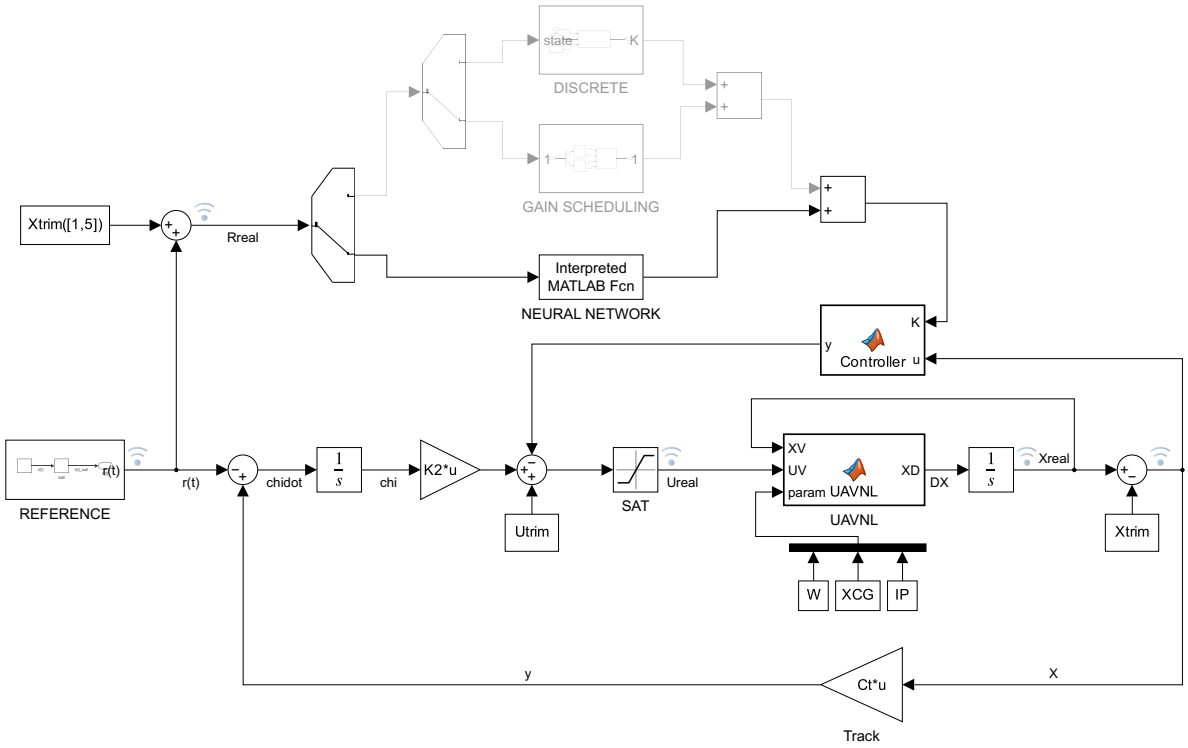


Figura C.5: `UAV_TRACK.slx`: Modelo completo de seguimiento.

Empleando el primer subsistema `DISCRETE` (detallado en la figura C.6) se estaría siguiendo la opción de elegir un controlador «discreto»: el que se definía por intervalos de operación, discretizando la envolvente de vuelo. Así, la variable `Vs_range` indica los saltos de los intervalos (los que en el ejemplo de la figura 4.13 de la sección 4.5 eran cada 5 m/s en el intervalo de 10 a 30 m/s) y `Ks_range` contiene las matrices del controlador de cada intervalo. La función `state2k_discrete` del modelo aparece en el listing A.5.

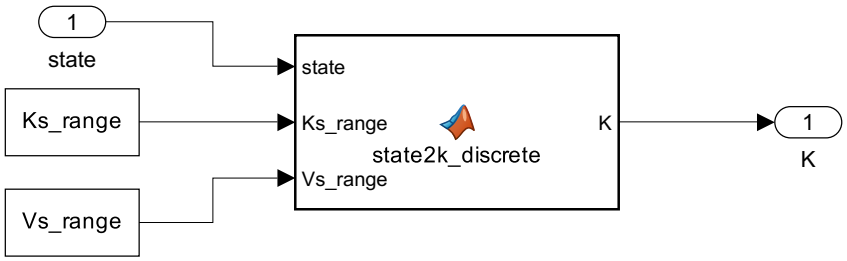


Figura C.6: `UAV_TRACK\sub1.slx`: Subsistema para el controlador discreto.

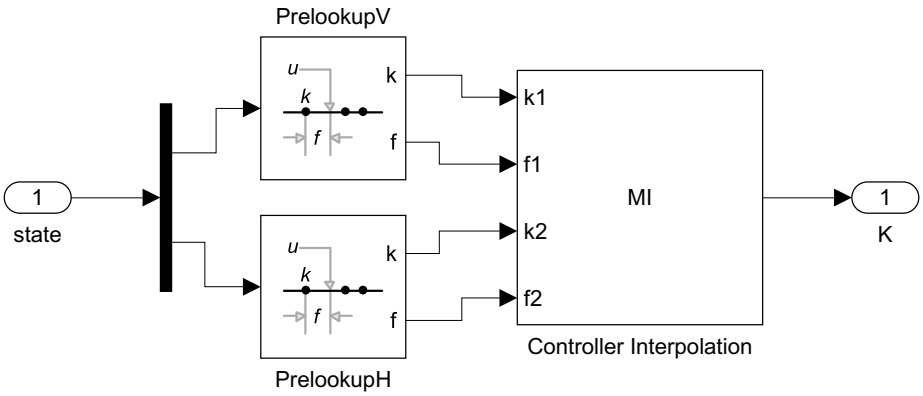


Figura C.7: `UAV_TRACK\sub2.slx`: Subsistema para el Gain Scheduling.

El sistema `GAIN SCHEDULING` es la segunda opción (figura C.7). Esta es la forma en la que se ha implementado la planificación de ganancias con bloques de Simulink. La entrada del estado se divide en sus dos componentes escalares y se hacen pasar por dos bloques `Prelookup`[1]que obtienen los valores necesarios para la posterior interpolación del controlador. Estos bloques tienen dentro los puntos de la malla donde se ha diseñado la familia de controladores (los puntos de las variables de planificación). Así, el primer bloque tiene los puntos asociados a la velocidad (los puntos de la malla de 10 a 30 m/s cada 5 m/s) y el segundo tiene los puntos de 100 a 3100 metros en intervalos de 500 m. Lo que devuelven estos bloques son el índice, k, y la fracción, f, replicando el funcionamiento de una tabla de valores. (La única diferencia es que la interpolación aquí es matricial, ya que se está interpolando toda la matriz K del controlador, a partir de las dos parejas (k, f) de estos dos bloques).

[1]Documentación de `Prelookup` en: https://es.mathworks.com/help/simulink/slref/prelookup.html.

El encargado de esta interpolación es el bloque «Controller Interpolation» (un bloque del tipo `Matrix Interpolation`[2]) que tiene dentro la *tabla* para interpolar dichas ganancias (que en realidad es un array de 4 dimensiones).

La tercera y última forma de escoger el controlador es mediante la red neuronal y la función a la que llama el bloque de «Interpreted MATLAB Function» bajo el nombre de `NEURAL NETWORK` en la figura del modelo C.5 se detalla en el listing A.6. Esta rama es la que emplea la inferencia de las ganancias a partir del estado de vuelo por medio de la red neuronal según lo descrito en la sección 4.3.

[2]Documentación de `Matrix Interpolation` en: https://es.mathworks.com/help/simulink/ref_extras/matrixinterpolation.html.

Bibliografía

Gómez Tierno, M. A., Pérez Cortés, M., and Puentes Márquez, C. (2012). *Mecánica del vuelo. 2nd*, volume 14. Garceta grupo editorial.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

NASA (2017). Naca airfoils.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

ßengel, Y. A., Cimbala, J. M., and Sknarina, S. F. (2006). *Mecánica de fluidos: fundamentos y aplicaciones*. McGraw-Hill Interamericana.

Stevens, B. L., Lewis, F. L., and Johnson, E. N. (2015). *Aircraft control and simulation: dynamics, controls design, and autonomous systems*. John Wiley & Sons.

The MathWorks Inc. (2006a). lqr. https://es.mathworks.com/help/control/ref/lti.lqr.html.

The MathWorks Inc. (2006b). trim. https://es.mathworks.com/help/simulink/slref/trim.html.

The MathWorks Inc. (2007). linmod. https://es.mathworks.com/help/simulink/slref/linmod.html.

Williams, R. L., Lawrence, D. A., et al. (2007). *Linear state-space control systems*. John Wiley & Sons.