UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Caminos, Canales y Puertos

Estimación de la trayectoria de aeronaves no tripuladas mediante machine learning.

Trabajo Fin de Máster

Máster Universitario en Sistemas Inteligentes de Transporte

AUTOR/A: Garrido Martínez, Javier

Tutor/a: Balbastre Tejedor, Juan Vicente

Cotutor/a: Vico Navarro, Joaquín

CURSO ACADÉMICO: 2023/2024

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE CAMINOS, CANALES Y PUERTOS

MUSIT: Máster Universitario en Sistemas Inteligentes de Transporte

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

“Estimación de la trayectoria en UAVs mediante Machine Learning.”

TRABAJO FINAL DE MÁSTER

Autor/a:
Javier Garrido Martinez

Tutor/a:
Juan Vicente Balbastre Tejedor
Joaquin Vico Navarro

VALENCIA, 2024

Abstract

In this work, we will develop a trajectory estimation model using neural networks in PyTorch so that, given the previous trajectory data, we will be able to produce an estimate of the aircraft trajectory for the following instants. For the generation of the dataset with which the model will be trained and tested, we will use the BB-Planner simulator of the BUBBLES aircraft ConOps capable of generating flight trajectories with different configurations. In addition, some modifications will be made to the trajectory model to find the best possible solution.

Keywords UAV, Machine Learning, neural networks, simulation, trajectory estimation.

Resumen

En este trabajo, desarrollaremos un modelo de estimación de trayectoria mediante redes neuronales en PyTorch para que, dados los datos de trayectoria anteriores, ser capaces de producir una estimación de la trayectoria de la aeronave para los instantes siguientes. Para la generación del dataset con el cual se entrenará y testeará al modelo, se hará uso del simulador BB-Planner del ConOps de aeronaves BUBBLES capaz de generar trayectorias de vuelo con distintas configuraciones. Además, se harán algunas modificaciones al modelo de trayectoria para encontrar la mejor solución posible.

Palabras clave UAV, Machine Learning, redes neuronales, simulación, estimación de la trayectoria.

Índice general

1. Objetivos **5**

2. Estado del Arte **7**
2.1. Concepto de Machine Learning 9
2.2. Proyecto BUBBLES (U-Space) 13
2.2.1. BB-Planner 14
2.3. Proyectos de investigación 18

3. Metodología **21**

4. Solución propuesta **25**
4.1. Generación datos trayectorias (BUBBLES BB-PLANNER) 26
4.2. Creación del dataset para el modelo 27
4.2.1. Extracción y procesamiento datos salida BUBBLES BB-PLANNER. . 27
4.3. Arquitectura del modelo 29
4.3.1. Entrenamiento y validación del modelo de ML 32

5. Conclusión **39**

A. Relación con los ODS de la agenda 2030 **41**

B. Bibliografía **45**

1. Objetivos

El objetivo principal del presente trabajo es desarrollar un sistema basado en Machine Learning (ML, Aprendizaje Automático) que sea capaz de estimar las trayectorias futuras de un UAV (Unmanned Aerial Vehicles, Vehículos Aéreos no Tripulados) basándose en sus trayectorias pasadas. Para ello, consideramos que deben de cumplirse los siguientes objetivos específicos que den lugar al objetivo principal:

1. Generar los conjuntos de datos suficientes sobre trayectorias de vuelo, mediante un simulador de trayectorias de UAVs (BB-Planner-BUBBLES-SJU), para un correcto funcionamiento del modelo de estimación de trayectoria.

2. Seleccionar la arquitectura de modelo de ML adecuada para el modelo de estimación de trayectoria.

3. Seleccionar la métrica adecuada para determinar el funcionamiento del modelo de estimación de trayectoria.

4. Seleccionar las características apropiadas de las trayectorias generadas por el simulador para el funcionamiento del modelo.

5. Implementar el modelo de estimación de trayectoria mediante la librería PyTorch de Python variando algunos de sus parámetros.

6. Analizar el rendimiento de los modelos mediante la métrica escogida para determinar posibles mejoras futuras.

2. Estado del Arte

Según la normativa europea, se entiende por UAS (comúnmente llamados drones o UAVs en inglés) cualquier aeronave que esté diseñada para operar de forma autónoma o para ser pilotada a distancia sin un piloto a bordo, así como el equipo necesario para controlarla de forma remota.Tiene consideración de aeronave funcional, por lo que su operación debe estar integrada con las operaciones del resto de usuarios del espacio aéreo[1].

Durante la última década, el sector de las aeronaves no tripuladas ha despertado un gran interés debido a su potencial y a la multitud de aplicaciones que ofrece, especialmente en entornos urbanos. Esto ha llevado a las autoridades y al público en general a plantear diferentes cuestiones y preocupaciones, entre las que la seguridad es la más preponderante [1].

Aunque los UAVs se utilizaron de forma significativa en algunas operaciones militares en los años 90, no fue hasta principios del siglo XXI cuando el interés por esta tecnología se disparó en el mundo civil. Hoy en día, existe una enorme expectación sobre una multitud de usos profesionales potenciales en muchos campos (movilidad urbana, reparto, seguridad pública, entretenimiento, agricultura, estudio de infraestructuras, etc.), así como en el ocio. Gran parte de ese mercado corresponde a actividades (movilidad, seguridad pública, reparto, etc.) que tendrán lugar dentro del espacio aéreo VLL (Very Low Level) sobre zonas urbanas densamente pobladas, así como sobre zonas industriales. Como resultado, se espera que para 2035 el total de horas de vuelo de los UAVs que vuelen en el VLL no controlado sobre zonas densamente pobladas supere el 75 % de las horas de vuelo del resto de la flota, incluidas las aeronaves tripuladas y no tripuladas que operan en el Cielo Único Europeo (CUE) controlado y no controlado. El uso de UAVs, como ocurre con casi cualquier actividad humana, conlleva algunos riesgos. Debido a este aumento en la actividad, tanto en horas de vuelo como en número de aeronaves sobre zonas densamente pobladas, es necesario un marco de seguridad para la operación de este tipo de dispositivos. Con ellos nace el concepto de U-Space, que engloba un conjunto de sistemas, servicios y procedimientos específicos que han sido diseñados para permitir el acceso seguro, eficiente y asequible al espacio aéreo de operaciones de UAVs numerosas o complejas, sobre la base de desarrollos técnicos con un alto grado de digitalización y automatización. Su propósito es lograr la gestión automatizada del tráfico de UAVs, especialmente en el espacio aéreo de baja altitud y en entornos urbanos, e integrarlo con seguridad con el sistema de gestión del tráfico aéreo ya existente para la aviación tripulada. El despliegue del U-Space desarrollará los elementos necesarios para la operación de drones en los llamados espacios aéreos U-Space, conforme a criterios de seguridad, protección, privacidad y medioambiente. Su despliegue será progresivo y requerirá la adecuada coordinación de un buen número de actores involucrados, tanto en la esfera pública como privada. El despliegue de U-Space trae consigo nuevos actores al espacio aéreo[2]:

[1] Ministerio de Transportes de España: UAVs (`www.transportes.gob.es/aviacion-civil/politica-espacio-aereo`)

[2] ENAIRE: U-Space (`www.enaire.es/servicios/drones`)

- el operador/piloto de drones que operará en entorno U-Space conectándose a un proveedor de servicio U-Space,
- proveedores de servicios U-Space (USSP): ENAIRE y otras organizaciones podrán certificarse para prestar los servicios U-Space a los operadores UAVs,
- proveedores de servicios de información común (CISP), que facilita la información necesaria para los USSPs. En este caso, España ha optado por la implantación de un modelo de prestación de servicio centralizado donde se designará a ENAIRE como proveedor único de estos servicios.

Los elementos claves de un U-Space son[3]:

- **Espacios aéreos U-Space**. Zonas geográficas de UAVs designadas por el Estado, en las que solo se permite que se lleven a cabo operaciones de UAVs con el apoyo de servicios U-Space.
- **Servicios obligatorios U-Space**. Se deben prestar obligatoriamente en los espacios aéreos U-Space que se designen.
  - **Identificación de Red**. Proporciona el número de registro de los operadores de UAVs, así como la información de ubicación, trayectoria y rumbo de los UAVs.
  - **Geoconsciencia**. Información sobre condiciones operacionales, limitaciones del espacio aéreo o restricciones temporales existentes.
  - **Autorización de Vuelo**. Garantiza que las operaciones que se realicen en un mismo volumen de espacio aéreo U-Space estén libres de conflicto con otros UAVs y zonas UAVs que puedan tener restricciones.
  - **Información de Tráfico**. Facilita información a los operadores de UAVs sobre otros tráficos, tripulados y no tripulados, que puedan encontrarse en las proximidades de sus aeronaves.
- **Servicios opcionales U-Space**. Se pueden prestar adicionalmente a los servicios obligatorios.
  - **Información Meteorológica**. Apoya las fases de planificación y ejecución del vuelo para mantener la seguridad.
  - **Supervisión de la Conformidad**. Avisa a los operadores que se desvían de la trayectoria planificada en su autorización de vuelo, así como a otros que operen en las proximidades de los UAVs afectados, a los proveedores de servicios U-Space y a los operadores ATS.
- **Proveedores de servicios**. Existen dos nuevos tipos de proveedores de servicios:

[3]Ministerio de Transportes de España: U-Space (`www.transportes.gob.es/aviacion-civil/politica-espacio-aereo/`)

- **Proveedores de servicios comunes de información (CISP)**. Difunden la información común necesaria para intercambiar datos operativos estáticos y dinámicos, siendo fuente única y confiable de toda la información común en los espacios aéreos U-Space bajo su responsabilidad.
- **Proveedores de servicios U-Space (USSP)**. Los servicios U-Space son prestados por USSPs certificados, durante todas las fases de operación.

En este contexto aparece BUBBLES, el cual presentamos en la sección 2.2.
Hasta hace unos años, los métodos usados para la gestión del tráfico aéreo consistían en algoritmos clásicos creados para modelizar aeronaves del tipo ala fija u otros tipos de aeronaves tripuladas como aeronaves de alas giratorias (helicópteros). Este tipo de aeronaves, principalmente debido a su tamaño por el hecho de ser tripuladas, tienen un comportamiento en vuelo muy distinto a las aeronaves no tripuladas. Los UAVs pueden ser de un tamaño mucho menor y más ligeros, proporcionando una maniobrabilidad muy superior a una aeronave tripulada. Además, al no llevar pasajeros (ni carga, en la mayoría de los casos), no tienen las limitaciones de fuerzas de aceleración que serian peligrosas para humanos y otros tipos de carga o las limitaciones de aceleración para el confort durante el vuelo de pasajeros de las aeronaves tripuladas. Otro factor a tener en cuenta es que, los pilotos de aeronaves no tripuladas tienen una formación muy distinta a los pilotos de aeronaves tripuladas (distintos tipos de maniobras, control remoto de la aeronave con distintos sistemas de percepción, mayores limitaciones en el tiempo de vuelo propiciados por la autonomía de estos sistemas...). En consecuencia de estos factores, los UAVs presentan una tipología de trayectorias muy distinta a las aeronaves tripuladas convencionales, lo que hace difícil adaptar los actuales sistemas de gestión de vuelo y análisis de trayectorias a este nuevo tipo de aeronave. En este contexto, la inteligencia artificial y el aprendizaje automático, presentan una posible mejora debido a su mayor flexibilidad para analizar trayectorias en vuelo con mayor incertidumbre y a su capacidad de generalizar su conocimiento sobre los datos entrenados con ella.

En los últimos años, el Machine Learning (ML) ha experimentado un amplio desarrollo, tanto a nivel de desarrollo y análisis de distintos modelos y la importancia de los datos utilizados para ellos, como en la amplitud de las áreas de investigación en las que se ha comenzado a aplicar. Este desarrollo viene dado por diversos factores tales como la mejora en la eficiencia computacional de los modelos de ML, y la mejora del hardware comercial de amplio uso, así como el cada vez mayor acceso a datos de todo tipo gracias a internet y los buenos resultados que están consiguiendo esta clase de algoritmos para tareas que, a priori, pueden ser muy difíciles de modelar a mano. Existen multitud de ejemplos de uso del aprendizaje automático en tareas de gestión aeroespacial y, en concreto, en modelización, gestión y estimación de la trayectoria entre distintas aeronaves; distintos investigadores han hecho uso de modelos de redes neuronales, aprendizaje por refuerzo o modelos basados en visión artificial para tratar este tipo de problemas, como describiremos con mayor detalle en la sección 2.3 Proyectos de investigación.

2.1. Concepto de Machine Learning

En general, el aprendizaje automático consiste en uno (o varios) modelos matemáticos que “aprenden” características de unos datos de entrada para proporcionar una salida en

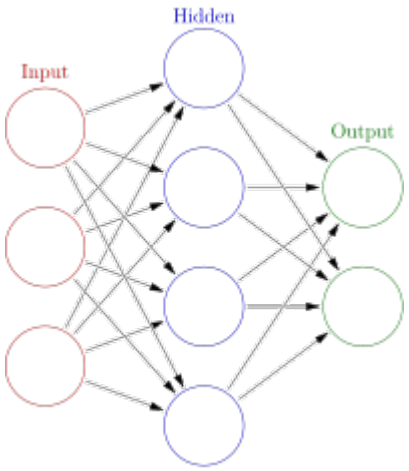


Figura 2.1.: Modelo de red neuronal artificial, la primera capa es la capa de entrada o input, la ultima capa es la capa de salida o output, las capas de la red entre estas dos capas de entrada/salida son las capas ocultas o hidden

base a ellos. Este aprendizaje, viene dado por la optimización de una serie de funciones matemáticas que actúan como métricas para establecer el error entre el resultado generado por el modelo y el resultado esperado (en el caso de modelos supervisados, el resultado esperado suele venir dado mediante una verdad anotada o ground-truth mientras que, en el caso de los modelos de aprendizaje por refuerzo, el proceso de aprendizaje se lleva a cabo generando resultados a partir del entrenamiento previo del modelo y comparando estos resultados con la métrica que indica la bondad de la solución, en nuestro caso, nos centramos en los modelos supervisados), estas métricas se conocen como función de coste o loss. Mediante el cálculo de este error, se pueden modificar iterativamente ciertos parámetros del modelo (los llamados parámetros entrenables), para que el modelo vaya reduciendo el margen entre los resultados generados y los esperados, este proceso se conoce como entrenamiento del modelo. Finalmente, se escoge un conjunto de datos que no haya sido usado en el proceso de entrenamiento del modelo para analizar la capacidad de generalización del modelo sobre unos datos sobre los que no ha sido entrenado, pero que conservan la estructura de los datos usados durante el entrenamiento. Por ello, en primer lugar, se escogen las características con las cuales se pretende conseguir el resultado de salida. A continuación, se escoge un modelo matemático que sea capaz de tratar estas características de entrada (árbol de decisión, redes neuronales…). Este modelo, ajustará los parámetros entrenables durante el entrenamiento para minimizar el error. Finalmente, se pueden modificar distintos parámetros del modelo para disminuir su error o aumentar su eficiencia. Una vez entrenado y testeado, el modo de uso del modelo consiste en introducir como entrada unos datos con la misma estructura y dimensionalidad que los datos de entrenamiento, estos datos serán usados en el interior del modelo para calcular el resultado. La salida dl modelo será el resultado de este cómputo, que conserva la dimensionalidad con las salidas que el modelo ha proporcionado durante la fase de entrenamiento. Este proceso de uso se denomina fase de inferencia. En la imagen 2.1 se muestra el modelo básico de red neuronal artificial. Profundizando en el modelo de neurona artificial [2], se dice que una neurona etiquetada como j, recibiendo una entrada $p_j(t)$ de

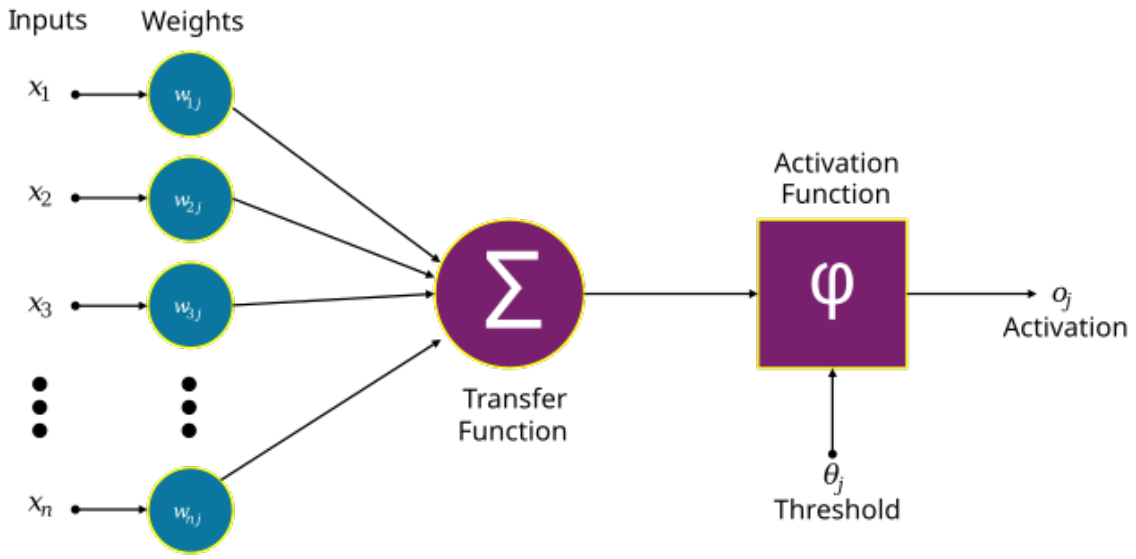


Figura 2.2.: Modelo de neuronal artificial

neuronas predecesoras, consiste en: Una activación $a_j(t)$, el estado de la neurona, en función de un parámetro de tiempo discreto, Un umbral opcional θ_j, que permanece fijo a no ser que se cambie durante el aprendizaje, Una función de activación f que calcula la nueva activación en un tiempo dado $t+1$ a partir de $a_j(t)$, θ_j, y la entrada de la red $p_j(t)$, lo que da lugar a la relación: $a_j(t+1) = f(a_j(t), p_j(t), \theta_j)$, Y una función de salida f_out que calcula la salida a partir de la activación $o_j(t) = f_out(a_j(t)$. Una neurona de entrada no tiene predecesora, pero sirve de interfaz de entrada para toda la red. Del mismo modo, una neurona de salida no tiene sucesora, por lo que sirve de interfaz de salida de toda la red. La función de propagación calcula la entrada $p_j(t)$ a la neurona j a partir de las salidas $o_i(t)$ y normalmente tiene la forma: $p_j(t) = \sum_i o_i(t) w_{ij}$ Se puede añadir un término de sesgo, cambiando la función de propagación a la siguiente: $p_j(t) = \sum_i o_i(t) w_{ij} + w_{0j}$, donde w_{0j} es un bias. En la imagen 2.2 se muestra el modelo de neurona artificial.

Durante el proceso de entrenamiento, se hace uso del método de backpropagation [3] mostrado en la imagen 2.3.

En ML, la retropropagación (backpropagation) es un método de estimación del gradiente utilizado habitualmente para entrenar redes neuronales con el fin de calcular las actualizaciones de los parámetros de la red. Es una aplicación eficiente de la regla de la cadena a las redes neuronales. Backpropagation calcula el gradiente de una función de pérdida con respecto a los pesos de la red para un único ejemplo de entrada-salida, y lo hace de forma eficiente, calculando el gradiente capa por capa, iterando hacia atrás desde la última capa para evitar cálculos redundantes de términos intermedios en la regla de la cadena. Este proceso de entrenamiento se ilustra en las imágenes 2.4 y 2.5

En el aprendizaje supervisado, una secuencia de ejemplos de entrenamiento $(x_1, y_1), \ldots, (x_p, y_p)$ produce una secuencia de pesos $w_0, w_1, \ldots, w_p$ partiendo de un peso inicial w_0, normalmente elegido al azar. Estos pesos se calculan a su vez de la siguiente forma: en primer lugar calcular w_i utilizando sólo (x_i, y_i, w_{i-1}) para $i = 1, \ldots, p$. La salida del algoritmo es entonces

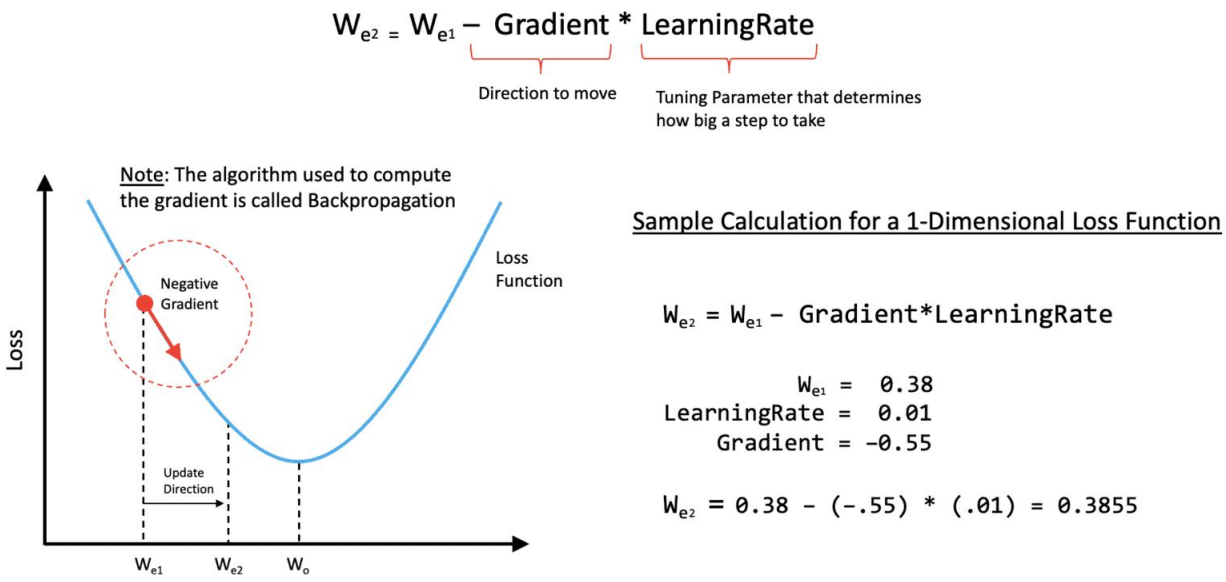


Figura 2.3.: Proceso de backpropagation[4].

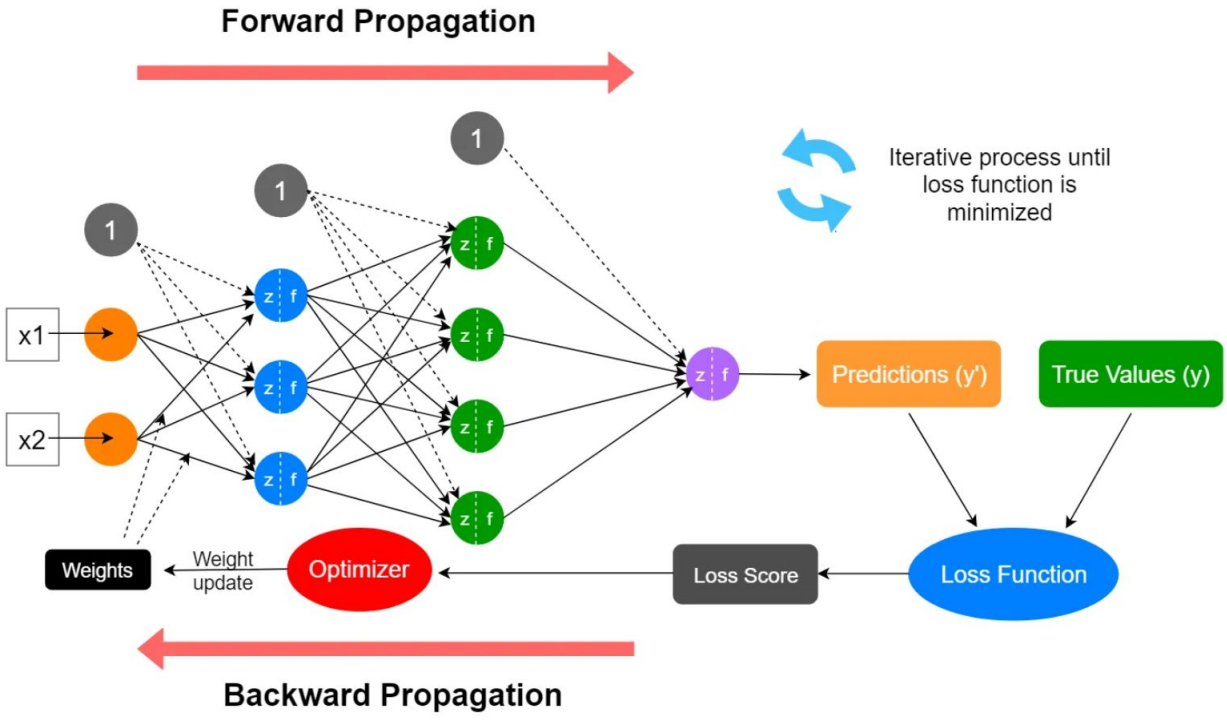


Figura 2.4.: Proceso general de entrenamiento de una red neuronal.

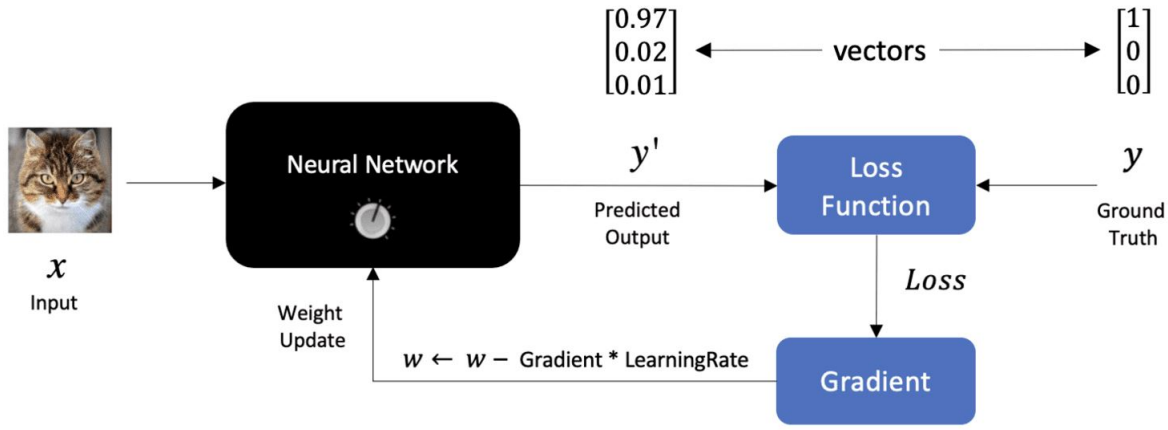


Figura 2.5.: Proceso de entrenamiento de una red neuronal para la tarea de clasificación.

w_p, dando una nueva función $x \mapsto f_N(w_p, x)$. El cálculo es el mismo en cada paso, por lo que sólo se describe el caso $i = 1$. w_1 se calcula a partir de (x_1, y_1, w_0) considerando un peso variable w y aplicando descenso gradiente a la función $w \mapsto E(f_N(w, x_1), y_1)$ para encontrar un mínimo local, comenzando en $w = w_0$. Esto hace que w_1 sea el peso minimizador para esa red, esos datos de entrenamiento y el peso inicial w_0, encontrado por descenso gradiente.

2.2. Proyecto BUBBLES (U-Space)

BUBBLES propone un concepto de operaciones (ConOps) para un Servicio de Gestión de Separación (SMS) proveído por el U-Space, cuyo objetivo es proveer a los usuarios (operadores y proveedores de servicios) la información requerida para garantizar que la capa de provisión de separación del proceso de gestión de conflictos se realiza homogéneamente y de acuerdo con un criterio especificado. BUBBLES es un proyecto financiado por SESAR Joint Undertaking (SJU) para una Investigación Exploratoria (ER) que tiene como objetivo definir un nuevo concepto de operaciones para proporcionar un servicio de gestión de separación en espacios aéreos de tipo U. La principal contribución de BUBBLES es proporcionar pautas para tratar la gestión de conflictos aéreos en la fase táctica, es decir, la fase de provisión de separación, en espacios aéreos de tipo U. En el contexto de BUBBLES, se presenta el concepto de servicio de gestión de la separación entre aeronaves, que aplica mínimos de separación dinámica adaptados a la clase de tráfico, el entorno operativo y la actuación del Sistema de Comunicación, Navegación y Vigilancia (CNS). Este concepto ha sido validado a través de vuelos de prueba en una zona rural como ambiente de simulación de su funcionamiento en un entorno U3, incluido una evaluación básica del desempeño humano (HPA). La resolución de conflictos en la aviación tripulada se caracteriza por la participación de seres humanos, el relativamente poco denso volumen de operaciones y la distribución centralizada de la provisión de los servicios de separación por los ATC (Air Traffic Control). Sin embargo, el U-Space presenta un escenario muy diferente, caracterizado por altos niveles de automatización, una mayor densidad de operaciones y la distribución de provisión de servicios en conjunto por varios USSPs (U-Space Service Providders). El proyecto BUBBLES ha sido testeado en vue-

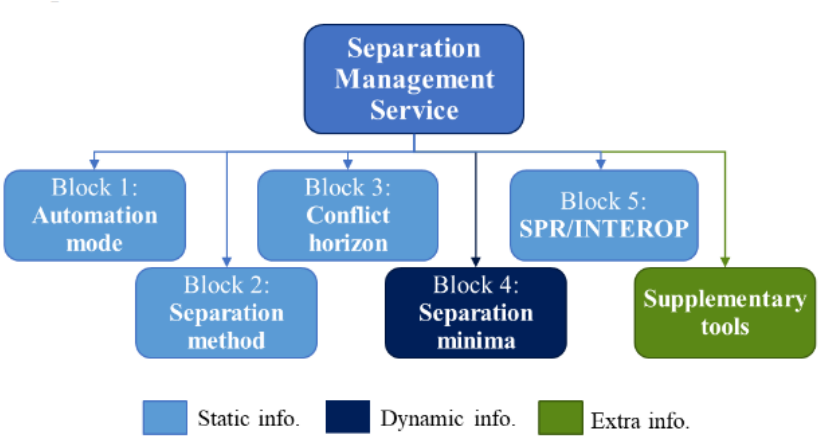


Figura 2.6.: Arquitectura conceptual del servicio de gestión de separación BUBBLES.

los de prueba reales con pilotos de drones, los cuales señalaron comentarios positivos sobre su funcionamiento, así como algunas posibles mejoras en su funcionamiento, como se puede constatar en [4] y [5]

2.2.1. BB-Planner

BB-Planner [6] es un software desarrollado por la UPV, que forma parte de las toolboxes:

1. BB-Planner-Tracker.
2. BB-Planner-Conflict.
3. BB-Planner-CC.

BB-Planner-Tracker

BB-Planner-Tracker: La toolbox BB-Planner-Tracker incorporada en el prototipo de herramienta BB-Planner 00.01.00:

1. Genera eficientemente trayectorias realistas, siguiendo la dinámica de UAVss multirotor y de ala fija.
2. Exporta las trayectorias generadas requeridas por las otras herramientas utilizadas en el Ejercicio de Validación BUBBLES 1:
   a) Planes de vuelo en formato JSON según la estructura requerida por el IBP 00.01.00.00 de INDRA-SIM.
   b) Trayectorias en formato de valores separados por comas (CSV) de acuerdo con la estructura requerida por el prototipo de herramienta BUBBLES AI Tool 00.01.00.

La toolbox BB-Planner-Tracker permite al usuario:

1. Elegir los UAVss que se utilizan para la misión que estamos generando.

2. Establecer una serie de waypoints que definan el despegue.
3. Seleccionar los puntos que limitan el área de la misión.
4. Seleccionar los waypoints en los que se producirá el aterrizaje.

Se pueden añadir los siguientes errores a la trayectoria para hacerla más realista:

1. Error técnico de vuelo y error del sistema de navegación para el error de posición,
2. probabilidad de detección,
3. error de detección durante un largo período de tiempo, y
4. errores de latencia.

BB-Planner-Conflict

La toolbox de BB-PLANNER-Conflict incorporada en el BB-Planner Tool prototype 00.01.00, es la encargada de:

1. calcular las frecuencias de conflicto para cualquier combinación de trayectorias definidas dentro de la misma operación teniendo en cuenta las medidas de mitigación estratégicas y tácticas.
2. Realizar este análisis de conflictos entre cada par de aeronaves. BB-Planner-Conflict permite seleccionar el conjunto de trayectorias a analizar, junto con los mínimos de separación aplicables para la separación vertical y horizontal.

BB-Planner-CC

La toolbox CC es un simulador Monte Carlo 3D que ejecuta trayectorias aleatorias simples (es decir, principalmente horizontales y lineales) en un escenario predefinido y cuenta el número de eventos de separación que se producen entre las aeronaves implicadas, diferenciando su gravedad según el modelo AIM definido por BUBBLES ConOp. En la imagen 2.7 podemos observar un ejemplo de trayectorias de referencia generadas por BB-PLANNER para realizar las pruebas de vuelo reales en distintas localizaciones. Las trayectorias fueron generadas para simular distintos propósitos mostrados en la tabla 2.1

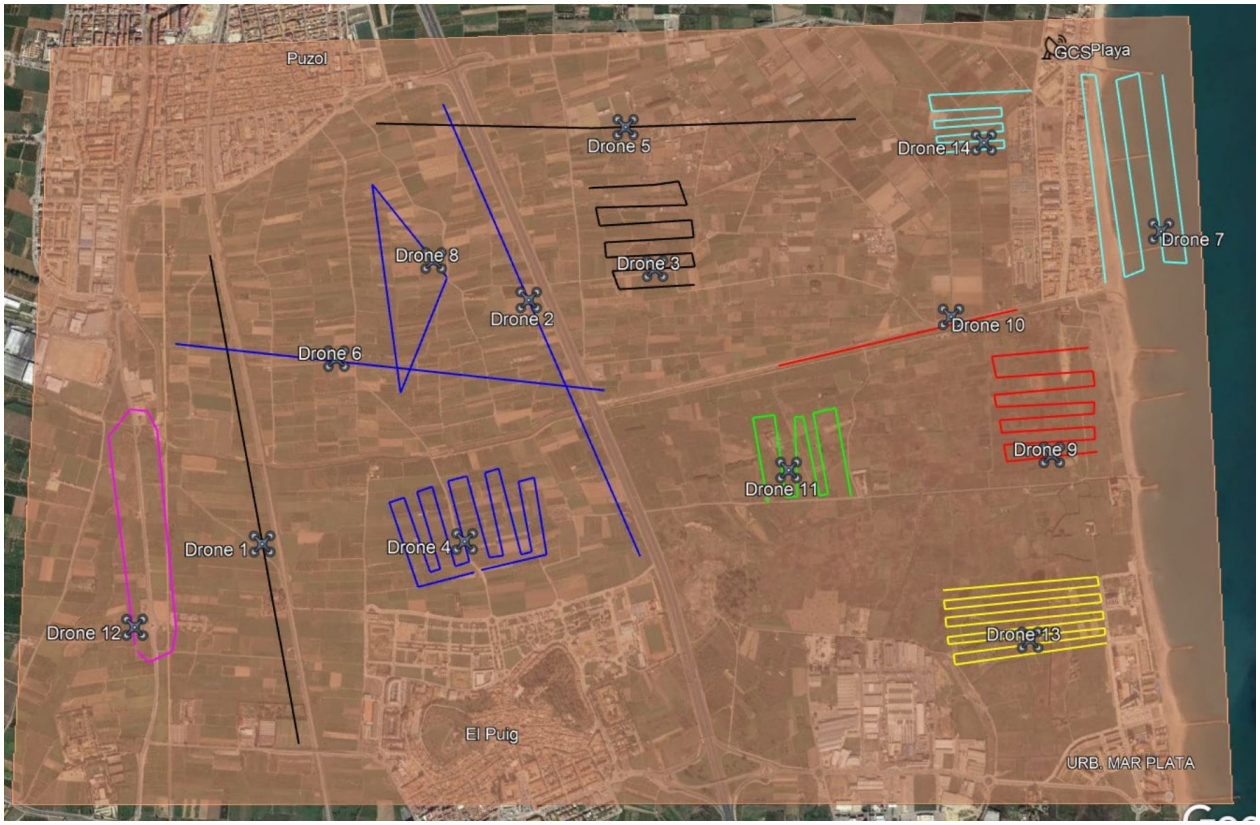


Figura 2.7.: Ejemplos de trayectorias de referencia generadas por BB-PLANNER para los test de vuelo reales.

Drone ID	Misión	Operador	Categoría Operacional	Drone
Drone 1	Inspección vías ferroviarias	UPV	STS-ES-02	DJI Matrice 300 RTK
Drone 2	Inspección vías automóviles	Policía de Valencia	STS-ES-02	DJI Mavic Enterp. Dual
Drone 3	Tareas agricultura	UPV	A3	DJI Mavic Enterp. Zoom
Drone 4	Tareas de vigilancia	Policía de Valencia	A3	DJI Mavic Enterp. Zoom
Drone 5	Entregas en ciudad	UPV	STS-ES-02	DJI Mavic Enterp. Zoom
Drone 6	Entregas en parque industrial	UPV	STS-ES-02	DJI Mavic Enterp. Dual
Drone 7	Vigilancia en playas	Policía de Benidorm	A3	DJI Mavic Enterp. Zoom
Drone 8	Agricultura de precisión	Policía de Valencia	STS-ES-02	DJI Mavic Enterp. Zoom
Drone 9	Vigilancia huertos para riesgo incendio	Bomberos Valencia	STS-ES-02	DJI Mavic Enterp. Advanced
Drone 10	Rescate animal	Bomberos Valencia	STS-ES-02	DJI Mavic Enterp. Advanced
Drone 11	Agricultura de precisión	AsDrone Spain	A3	DJI Phantom 4 PRO
Drone 12	Tareas de vigilancia	UAV Works	A3	Valak Patrol
Drone 13	Tareas agricultura	ASD drones	A3	DJI Mavic Enterp. Advanced
Drone 14	Fotogrametría	Policía de Benidorm	A3	DJI Mavic Enterp. Advanced

Cuadro 2.1.: Misiones realizadas durante las pruebas de vuelo.

2.3. Proyectos de investigación

Existen numerosos estudios sobre la predicción de trayectorias aéreas, tanto en aeronaves tripuladas convencionales, como en UAVs, tanto autónomos como controlados a distancia. Los usos en los que destaca esta tecnología van desde la estimación de la ocupación futura de un espacio aéreo determinado para su gestión eficiente y segura, hasta la pura predicción de la trayectoria en distintas aeronaves como necesidad para su manejo autónomo, pasando por sistemas recomendadores para la evitación y resolución de conflictos de vuelo. Por ejemplo, en el artículo Air Traffic Control Using Message Passing Neural Networks and Multi-Agent Reinforcement Learning [7] (Control de tráfico aéreo mediante redes neuronales de paso de mensajes y aprendizaje por refuerzo multiagente), los autores desarrollan una herramienta de recomendación para los controladores aéreos (ATCs) basada en redes neuronales que minimiza las trayectorias ineficientes, el número de instrucciones necesarias que deben de tomar los ATCs, los conflictos en vuelo y las alertas. Además, el protocolo utilizado permite llegar a un consenso entre los vuelos antes de tomar la mejor decisión conjunta, lo que incrementa la eficiencia de las trayectorias y penaliza las situaciones de riesgo como la pérdida de distancia de seguridad entre aeronaves.

Otro estudio donde se optimizan las funciones de ATC mediante ML es Machine learning-based flight trajectories prediction and air traffic conflict resolution advisory [8] (Predicción de trayectorias de vuelo basada en aprendizaje automático y asesoramiento sobre resolución de conflictos de tráfico aéreo). En esta publicación, se investigan las aplicaciones de ML en la detección de conflictos de vuelo y su resolución mediante Reinforcement Learning (aprendizaje por refuerzo). El modelo, alcanza altas tasas de éxito en los escenarios que se plantean de un contexto multi-agente; aunque los autores reconocen que sigue estando lejos de una aplicación operativa en un escenario real debido a la alta demanda de seguridad en la industria del tráfico aéreo, si puede ser usado con éxito como ayuda en la formación de nuevos ATCs.

Un estudio similar llevado a cabo en el espacio aéreo de la terminal de los aeropuertos es Hybrid Machine Learning and Estimation-Based Flight Trajectory Prediction in Terminal Airspace [9] (Predicción híbrida de trayectorias de vuelo basada en el aprendizaje automático y estimación en el espacio aéreo de la terminal).

Entre los estudios que se centran en la predicción de trayectoria para UAVs para distintos escenarios y no únicamente buscando su aplicación para ATC, podemos encontrar UAV Trajectory Modeling Using Neural Networks [10] (Modelado de trayectorias de UAV mediante redes neuronales). En este trabajo de la NASA se presenta una solución basada en redes neuronales para modelar la trayectoria de pequeños UAVs, sin necesidad de conocer su sistema de control o sus parámetros aerodinámicos, ya que son variables que pueden variar ampliamente entre aparatos manufacturados por las distintas empresas de fabricación de UAVs, además de ser sistemas cerrados en el caso de los sistemas de control. En este trabajo se usa una red neuronal con 9 dimensiones de entrada, 20 neuronas ocultas y 6 dimensiones de salida, entrenada y testada mediante datos generados por un modelo matemático del vehículo, con la que se predicen las trayectorias 4D.

Otro artículo donde los autores proponen sistemas de ML para la predicción de trayectoria es Short-term 4D Trajectory Prediction Using Machine Learning Methods [11] (Predicción de trayectorias 4D a corto plazo mediante métodos de aprendizaje automático), en este caso, se usa PCA (Principal Component Analysis) para reducción de dimensionalidad y las trayec-

torias se clasifican mediante un método de clustering para la entrada de la red neuronal.

Así como, en el artículo Trajectory prediction of UAV Based on LSTM [12] (Predicción de trayectoria de UAV basada en LSTM), los autores hacen uso de una red LSTM y las 4 posiciones anteriores del UAV para la predicción de su trayectoria.

Otro artículo donde se intenta predecir la trayectoria a corto plazo con las coordenadas anteriores es Short-Term 4D Trajectory Prediction for UAV Based on Spatio-Temporal Trajectory Clustering [13] (Predicción de trayectoria a corto plazo en 4D para UAV basada en la agrupación de trayectorias espaciotemporales). En este artículo, en primer lugar, se hace uso de una red neuronal (Red Neuronal Convolucional, CNN) para clasificar el tipo de trayectoria durante una ventana temporal para, a continuación, estimar las coordenadas en 4D en una predicción con rango de 0-3 segundos con una red LSTM.

Un enfoque similar se da en An Aircraft Trajectory Prediction Method Based on Trajectory Clustering and a Spatiotemporal Feature Network [14] (Método de predicción de la trayectoria de una aeronave basado en la agrupación de trayectorias y en una red de características espaciotemporales), donde se hace uso de una red CNN para la extracción de características de la trayectoria, y una red BiLSTM (Bidirectional Long Short-Term Memory) para obtener la estimación.

En el caso de A Generalized Approach to Aircraft Trajectory Prediction via Supervised Deep Learning [15] (Un enfoque generalizado para la predicción de trayectorias de aeronaves mediante aprendizaje profundo supervisado), los autores de la NASA realizan un estudio con distintas arquitecturas y tipos de datos de trayectoria para su predicción. Además, hacen uso de datos meteorológicos de distintas fuentes para, con todos estos distintos tipos de datos, intentar generalizar los modelos a distintos tipos de aeronaves, rutas de vuelo y condiciones ambientales.

En el artículo Real-time unmanned aerial vehicle flight path prediction using a bi-directional long short-term memory network with error compensation [16] (Predicción en tiempo real de la trayectoria de vuelo de un vehículo aéreo no tripulado mediante una red bidireccional de memoria a corto plazo con compensación de errores), los autores utilizan una red neuronal Bi-LSTM para la predicción de trayectoria de aeronaves, junto con un modelo matemático para la compensación de errores para la transformación de los datos de salida de la red al sistema Bessel geodésico.

Además, existen soluciones para generar datos de trayectorias artificiales mediante simuladores con el fin de ser utilizados en modelización de trayectorias para estos modelos de ML como el usado en Generating Synthetic Training Data for Deep Learning-Based UAV Trajectory Prediction [17] (Generación de datos de entrenamiento sintéticos para la predicción de trayectorias de UAV basada en aprendizaje profundo).

Desde la perspectiva de los dos elementos clave de la planificación global y local, el articulo UAV Formation Trajectory Planning Algorithms: A Review [18] (Algoritmos de planificación de trayectorias de formación de UAV: Una revisión), propone un marco para los algoritmos de planificación de trayectorias de formación de UAV, clasifica exhaustivamente los diferentes tipos de algoritmos y describe los diferentes tipos de algoritmos y sus variantes de forma unificada. Además, se lleva a cabo una revisión y un análisis estadístico sobre la base de la clasificación.

En Machine Learning for Aircraft Trajectory Prediction: a Solution for Pre-tactical Air Traffic Flow Management [19] (Aprendizaje Automático para la predicción de la trayectoria de

aeronaves: una solución para la gestión pretáctica del flujo de tráfico aéreo), se desarrolla una solución completa para la predicción de trayectoria en aeronaves. En este trabajo, el autor se centra en la fase pretáctica de la gestión de demanda y capacidad de tráfico del espacio aéreo, que abarca los 6 días previos al día de operaciones. Durante esta fase, la información por parte de EUROCONTROL del plan de vuelo es muy limitada o prácticamente nula, restringiéndose a las llamadas "Intenciones de vuelo", consistentes principalmente en los horarios. El modelo intenta predecir, mediante planes de vuelo anteriores por parte de la misma aeronave, las rutas que van a seguir los aeroplanos, mejorando al actual sistema PREDICT usado por EUROCONTROL.

3. Metodología

En general, para la tarea de predicción de trayectoria existen 2 metodologías, por un lado las basadas en algoritmos clásicos, como mediante el uso de filtros de Kalman [20] o mediante modelación física [21, 22] y por el otro, las basadas en modelos de inteligencia artificial.

La ventaja de los modelos clásicos como los filtros de Kalman es que su coste computacional generalmente es menor que entrenar una red neuronal, sin embargo, su desventaja es que, dependiendo de los modelos de redes neuronales y de filtros, la inferencia de una red neuronal puede ser computacionalmente igual o más eficiente a largo plazo que los filtros de Kalman. Por otro lado, la ventaja de los modelos basados en físicas es una mayor precisión en la estimación de la trayectoria a muy corto plazo, ya que se suelen modelizar fenómenos que tienen mucho impacto en la trayectoria. No obstante, sus principales desventajas son la dificultad de modelar los fenómenos físicos, su alto coste computacional, su divergencia a largo plazo de la trayectoria si no se incorporan mecanismos de corrección y su difícil generalización a otro vehículo del que las físicas puedan ser distintas al vehículo modelizado. Las principales ventajas del ML son su capacidad para extraer relaciones complejas de los datos de entrada, especialmente útiles cuando existen relaciones de alto nivel difíciles de modelar a mano. Estos modelos son capaces de manejar múltiples variables de entrada y de salida, así como modelar complejas relaciones no lineales. Además, ofrece gran capacidad de inferencia en los resultados, lo que lo hace altamente adaptable para una tarea donde es probable que existan errores o incluso ausencia de algunos datos en determinados momentos (debido a errores de transmisión o de cálculo de la posición o velocidad). Las principales desventajas del uso de ML son la necesidad de disponer de datos en cantidad y calidad sobre el problema a modelizar. La explicabilidad del modelo puede llegar a ser compleja, aunque se puede acotar mediante experimentación y el uso de las métricas correctas. Y, por último, el coste computacional (especialmente de entrenamiento, aunque también puede serlo de inferencia), puede llegar a ser muy alto en determinados modelos. Otra de las posibles desventajas es el sobre entrenamiento del modelo con los datos de entrenamiento, lo que puede hacerlo difícil de generalizar a otros datos [23, 24, 25].

En nuestro caso, debido a los distintos tipos de UAVs que existen, junto con la variedad de trayectorias que pueden adoptar, decidimos usar un modelo de redes neuronales para realizar la tarea de la predicción de trayectoria.

Para la programación del modelo, se utilizará el lenguaje de programación Python[1]. Python es un lenguaje de programación interpretado, interactivo y orientado a objetos. Incorpora módulos, excepciones, tipado dinámico, tipos de datos dinámicos de muy alto nivel y clases. Soporta múltiples paradigmas de programación más allá de la programación orientada a objetos, como la programación procedimental y funcional. Python combina una potencia notable con una sintaxis muy clara. Dispone de interfaces para muchas llamadas al sistema y bibliotecas, así como para varios sistemas de ventanas, y es extensible en C o C++. También

[1]Python: (`www.python.org/`)

puede utilizarse como lenguaje de extensión para aplicaciones que necesiten una interfaz programable. Por último, Python es portable: funciona en muchas variantes de Unix, incluidos Linux y macOS, y en Windows[2]. Python es un lenguaje de programación potente y fácil de aprender. Posee estructuras de datos eficientes de alto nivel y un enfoque sencillo pero eficaz de la programación orientada a objetos. La sintaxis elegante de Python y su tipado dinámico, junto con su naturaleza interpretada, lo convierten en un lenguaje ideal para la creación de scripts y el desarrollo rápido de aplicaciones en muchos ámbitos y en la mayoría de las plataformas. El intérprete de Python y la extensa biblioteca estándar están disponibles gratuitamente para las principales plataformas en el sitio web de Python, y pueden distribuirse libremente. El mismo sitio contiene también distribuciones y enlaces a muchos módulos, programas y herramientas Python de terceros, así como documentación adicional. El intérprete de Python se amplía fácilmente con nuevas funciones y tipos de datos implementados en C o C++ (u otros lenguajes invocables desde C). Python también es adecuado como lenguaje de extensión para aplicaciones personalizables[3].

Elegimos Python ya que es un lenguaje ampliamente usado en ciencia de datos, así como en investigación de modelos de aprendizaje automático. Debido a esto, Python disfruta de un amplio ecosistema de librerías para el modelado de sistemas de inteligencia artificial como Scikit Learn, Pytorch, Tensorflow o Keras así como de manipulación y representación de datos como Numpy, Pandas o Matplotlib. En nuestro caso, se utilizará la librería Pandas para generar el dataset a partir de la salida del BB-PLANNER, y para hacer las transformaciones necesarias a los datos para ser introducidos en la red neuronal. En el caso del modelo de trayectoria, se utilizará la librería Pytorch, ya que permite la suficiente personalización en el modelado de la arquitectura de la red neuronal.

Pandas es una herramienta de análisis y manipulación de datos de código abierto rápida, potente, flexible y fácil de usar, construida sobre el lenguaje de programación Python. Algunas de sus características más destacables son: Un objeto DataFrame rápido y eficaz para la manipulación de datos con indexación integrada, herramientas de lectura y escritura de datos entre estructuras de datos en memoria y distintos formatos reestructuración y pivotado flexible de conjuntos de datos, división inteligente basada en etiquetas, indexación sofisticada y subconjunto de grandes conjuntos de datos, agregar o transformar datos con un potente motor group by que permite operaciones de división, aplicación y combinación en conjuntos de datos, fusión y unión de conjuntos de datos de alto rendimiento, funcionalidad de series temporales: generación de intervalos de fechas y conversión de frecuencias, estadísticas de ventanas móviles, desplazamiento de fechas y desfase. Incluso crear desfases temporales específicos del dominio y unir series temporales sin perder datos. Altamente optimizado para el rendimiento, con rutas de código críticas escritas en Cython o C. Python con Pandas se utiliza en una gran variedad de ámbitos académicos y comerciales, como las finanzas, la neurociencia, la economía, la estadística, la publicidad, la analítica web, etc[4].

Un DataFrame es una estructura de datos bidimensional que puede almacenar datos de diferentes tipos (incluyendo caracteres, enteros, valores de coma flotante, datos categóricos y más) en columnas. Es similar a una hoja de cálculo, una tabla SQL o el data.frame de R[5].

[2]Python Docs: (`docs.python.org/3/faq/general.html`)
[3]Python Docs: (`docs.python.org/3/tutorial/index.html`)
[4]Pandas: (`pandas.pydata.org/`)
[5]Pandas: (`pandas.pydata.org/docs/getting_started/intro_tutorials/01_table_oriented.html`)

Se puede considerar como un contenedor tipo dict para objetos Series, la principal estructura de datos de pandas[6].

Pytorch[7] es un framework end-to-end de aprendizaje automático. PyTorch permite una experimentación rápida y flexible y una producción eficiente a través de un front-end fácil de usar, formación distribuida y un ecosistema de herramientas y bibliotecas. Algunas de sus características principales son[8]:

- Usado en producción: con TorchScript, PyTorch proporciona facilidad de uso y flexibilidad en el modo eager, además de la transición sin problemas al modo gráfico para la velocidad, la optimización y la funcionalidad en entornos de ejecución C++.
- TorchServe: es una herramienta fácil de usar para desplegar modelos PyTorch a escala. Es agnóstica a la nube y al entorno y soporta características como el servicio multimodelo, registro, métricas y la creación de puntos finales RESTful para la integración de aplicaciones.
- Entrenamiento distribuido: Optimiza el rendimiento tanto en investigación como en producción aprovechando el soporte nativo para la ejecución asíncrona de operaciones colectivas y la comunicación entre pares accesible desde Python y C++.
- Mobile (Experimental): PyTorch soporta un flujo de trabajo de extremo a extremo desde Python hasta el despliegue en iOS y Android. Amplía la API de PyTorch para cubrir tareas comunes de preprocesamiento e integración necesarias para incorporar ML en aplicaciones móviles.
- Ecosistema robusto: Una activa comunidad de investigadores y desarrolladores ha creado un ecosistema de herramientas y bibliotecas para ampliar PyTorch y apoyar el desarrollo en áreas que van desde la visión por ordenador hasta el aprendizaje por refuerzo.
- Compatibilidad nativa con ONNX: Pytorch es capaz de exportar modelos en el formato estándar ONNX (Open Neural Network Exchange) para acceder directamente a plataformas, tiempos de ejecución y visualizadores compatibles con ONNX, entre otros.
- Interfaz C: La interfaz C++ es una interfaz C++ pura para PyTorch que sigue el diseño y la arquitectura de la interfaz Python establecida. Está pensado para permitir la investigación en aplicaciones C++ de alto rendimiento, baja latencia y bare metal.
- Soporte en la nube: PyTorch está bien soportado en las principales plataformas en la nube, proporcionando un desarrollo sin fricciones y un fácil escalado a través de imágenes pre-construidas, entrenamiento a gran escala en GPUs, capacidad de ejecutar modelos en un entorno a escala de producción, etc.

En cuanto a las estructuras de datos, se utilizarán Dataframes de la librería Pandas, tanto para la carga de datos en memoria del fichero CSV del BBPLANNER, como para su tratamiento e introducción en la red neuronal durante el entrenamiento y el test. Cada registro

[6]Pandas: (`pandas.pydata.org/docs/reference/api/pandas.DataFrame.html`)
[7]PyTorch: (`pytorch.org/`)
[8]PyTorch: (`pytorch.org/features/`)

del dataframe, contendrá los datos necesarios asociados a un UAV en un momento dado, para el entrenamiento e inferencia de las trayectorias por la red. Para introducir los datos al modelo, se utilizará un objeto DataLoader[9] de Pytorch, ya que es el mecanismo estándar de esta librería para entrenar los modelos de redes neuronales, además de que nos permite definir una serie de parámetros para el modelo como el Batch Size, el número de hilos de ejecución para la carga de datos o evitar copias innecesarias de la estructura de datos en el dispositivo CUDA. La salida del modelo consiste en un tensor con las coordenadas estimadas a partir de las coordenadas anteriores introducidos en la entrada del modelo. En nuestro caso, elegimos un modelo LSTM ya que es un tipo de red neuronal que conserva cierta memoria en su contexto, lo que puede ser útil para tener en cuenta datos de trayectorias anteriores, además, ha mostrado tener buenos resultados en la bibliografía del estado del arte consultada, y es un tipo de modelo muy generalizado y utilizado en numerosos casos de uso de ML, por ello tiene la ventaja de estar ampliamente implementado de forma eficiente en la mayoría de librerías de redes neuronales y existe amplia documentación sobre su uso.

[9]PyTorch DataLoader: (`pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader`)

4. Solución propuesta

En primer lugar, para la estimación de la trayectoria por parte de nuestro modelo, definimos la trayectoria de un UAV mediante sus tres coordenadas espaciales (x, y, h), y su coordenada temporal t. Matemáticamente, esto es una matriz con dimensiones $3 \times m$, siendo m el número de trayectorias a estimar:

$$\begin{bmatrix} x_t & y_t & h_t \\ x_{t+1} & y_{t+1} & h_{t+1} \\ \vdots & \vdots & \vdots \\ x_m & y_m & h_m \end{bmatrix} \tag{4.1}$$

Computacionalmente, las trayectorias se han cargado en un Pytorch.Tensor que se explicará en adelante.
La solución se ha implementado en un portátil Lenovo Legion 5 15ITH6H con la siguiente configuración:
Sistema Operativo: Windows 10.0.19045
CPU: Intel i7-11800H
RAM: 32GB.
GPU: NVIDIA GeForce RTX 3070 Laptop GPU.
En primer lugar, mostramos el flujo de datos y ejecución, que viene dado por el diagrama de 4.1.

El flujo de datos y ejecución del modelo es el siguiente, los datos de trayectorias de los que disponemos son los generados por el simulador BB-PLANNER en formato CSV. Estos

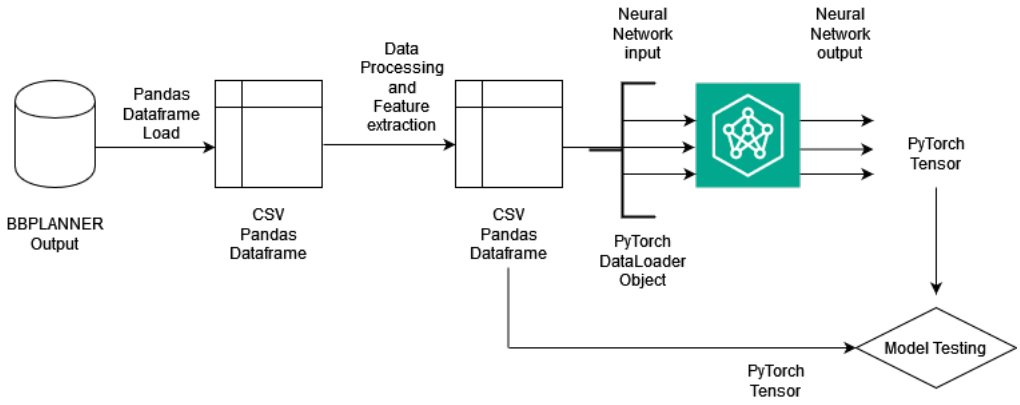


Figura 4.1.: Diagrama de flujo de datos de la solución.

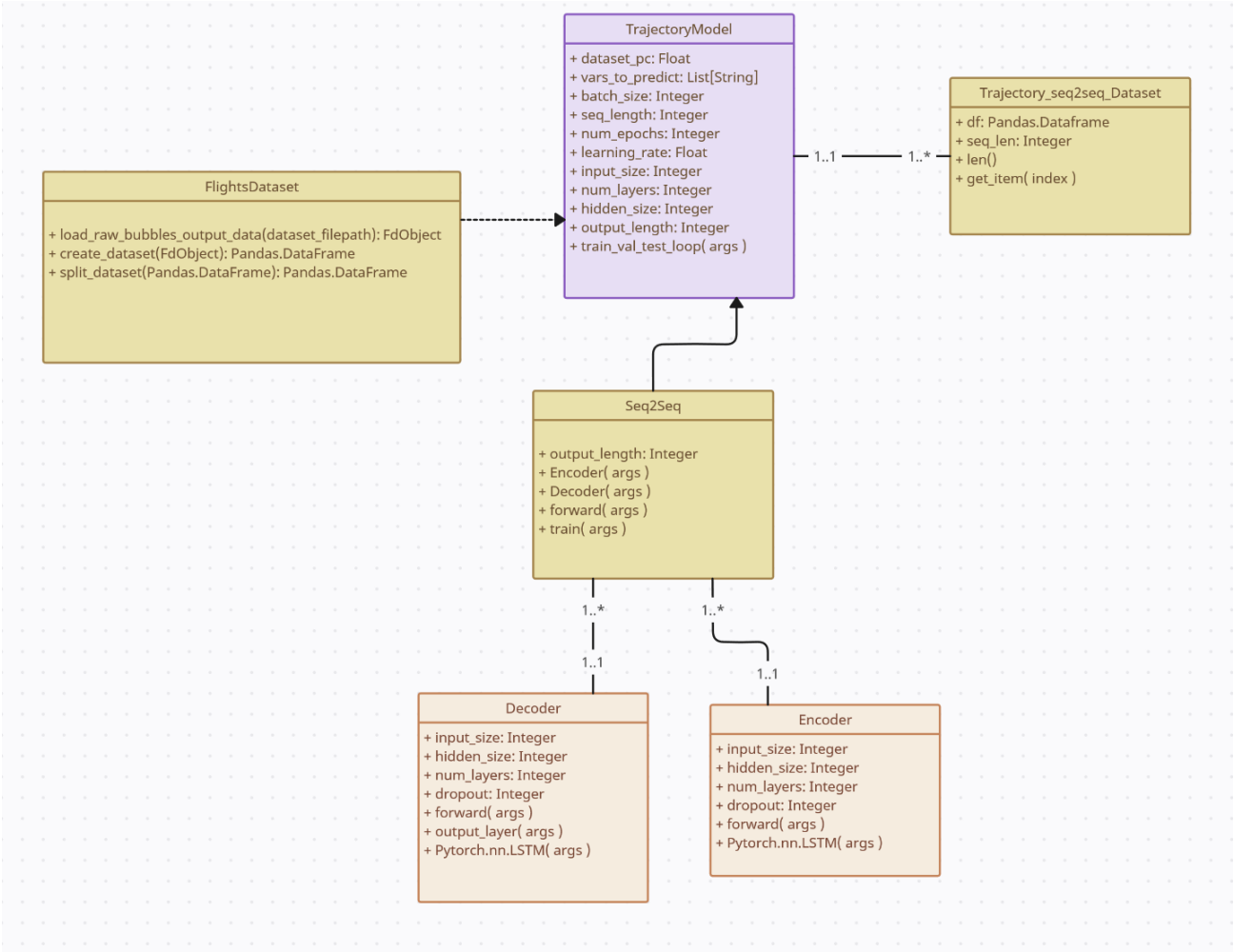
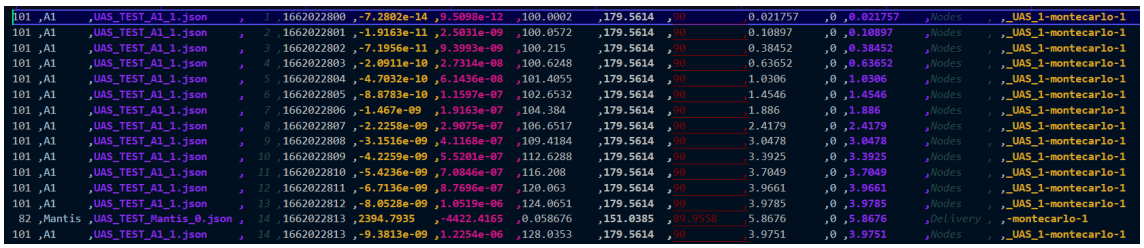


Figura 4.2.: Diagrama UML de la solución.

datos son cargados mediante Pandas en un Pandas Dataframe en memoria principal para su procesamiento. Una vez procesados, se cargan en un objeto Pytorch DataLoader que los convierte en un formato valido para ser usado en el entrenamiento del modelo. A continuación, se entrena el modelo con el Pytorch DataLoader y, finalmente, se extrae la salida para calcular el MSE (Mean Square Error) del conjunto de entrenamiento y validación, y realizar el testeo del modelo.

4.1. Generación datos trayectorias (BUBBLES BB-PLANNER)

Para la generación de las trayectorias artificiales que serán usadas en el entrenamiento y test de la red, se hace uso del módulo BB-PLANNER de BUBBLES. BBPLANNER genera trayectorias artificiales de UAVs a partir de un fichero de configuración JSON, donde se pueden especificar las características tanto de la simulación que realiza (paso temporal, configuraciones del ruido a aplicar sobre los datos, formatos de salida de los datos, numero



101	A1	UAS_TEST_A1_1.json	1	1662022800	-7.2802e-14	9.5098e-12	100.0002	179.5614	90	0.021757	0	0.021757	Nodes	_UAS_1-montecarlo-1	
101	A1	UAS_TEST_A1_1.json	2	1662022801	-1.9163e-11	2.5031e-09	100.0572	179.5614	90	0.10897	0	0.10897	Nodes	_UAS_1-montecarlo-1	
101	A1	UAS_TEST_A1_1.json	3	1662022802	-7.1956e-11	9.3993e-09	100.215	179.5614	90	0.38452	0	0.38452	Nodes	_UAS_1-montecarlo-1	
101	A1	UAS_TEST_A1_1.json	4	1662022803	-2.0911e-10	2.7314e-08	100.6248	179.5614	90	0.63652	0	0.63652	Nodes	_UAS_1-montecarlo-1	
101	A1	UAS_TEST_A1_1.json	5	1662022804	-4.7032e-10	6.1436e-08	101.4055	179.5614	90	1.0306	0	1.0306	Nodes	_UAS_1-montecarlo-1	
101	A1	UAS_TEST_A1_1.json	6	1662022805	-8.8783e-10	1.1597e-07	102.6532	179.5614	90	1.4546	0	1.4546	Nodes	_UAS_1-montecarlo-1	
101	A1	UAS_TEST_A1_1.json	7	1662022806	-1.467e-09	1.9163e-07	104.384	179.5614	90	1.886	0	1.886	Nodes	_UAS_1-montecarlo-1	
101	A1	UAS_TEST_A1_1.json	8	1662022807	-2.2258e-09	2.9075e-07	106.6517	179.5614	90	2.4179	0	2.4179	Nodes	_UAS_1-montecarlo-1	
101	A1	UAS_TEST_A1_1.json	9	1662022808	-3.1516e-09	4.1168e-07	109.4184	179.5614	90	3.0478	0	3.0478	Nodes	_UAS_1-montecarlo-1	
101	A1	UAS_TEST_A1_1.json	10	1662022809	-4.2259e-09	5.5201e-07	112.6288	179.5614	90	3.3925	0	3.3925	Nodes	_UAS_1-montecarlo-1	
101	A1	UAS_TEST_A1_1.json	11	1662022810	-5.4236e-09	7.0846e-07	116.208	179.5614	90	3.7049	0	3.7049	Nodes	_UAS_1-montecarlo-1	
101	A1	UAS_TEST_A1_1.json	12	1662022811	-6.7136e-09	8.7696e-07	120.063	179.5614	90	3.9661	0	3.9661	Nodes	_UAS_1-montecarlo-1	
101	A1	UAS_TEST_A1_1.json	13	1662022812	-8.0528e-09	1.0519e-06	124.0651	179.5614	90	3.9785	0	3.9785	Nodes	_UAS_1-montecarlo-1	
82	Mantis	UAS_TEST_Mantis_0.json	14	1662022813	2394.7935	-4422.4165	0.058676	151.0385	89.9558	5.8676	0	5.8676	Delivery	-montecarlo-1	
101	A1	UAS_TEST_A1_1.json	14	1662022813	-9.3813e-09	1.2254e-06	128.0353	179.5614	90	3.9751	0	3.9751	Nodes	_UAS_1-montecarlo-1	

Figura 4.3.: Ejemplo de los primeros 15 registros de la salida proporcionada por BB-PLANNER en CSV.

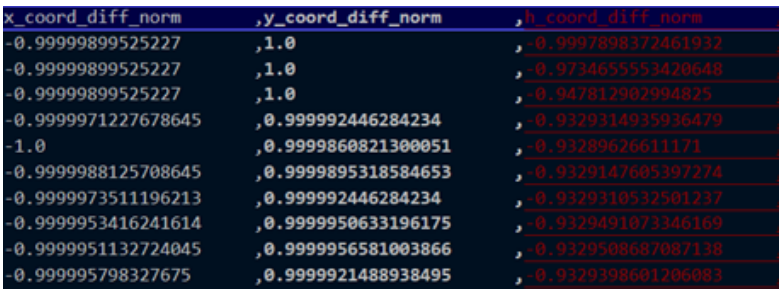
de iteraciones de la simulación...), como de atributos de los UAVs tales como el nombre del UAV, la categoría de riesgo, tipo de UAV, velocidades, alturas y ángulos de giro máximos y mínimos, coordenadas de despegue y aterrizaje y waypoints por los que debe de pasar el UAV. Como resultado de la simulación, BB-PLANNER genera un fichero CSV con los datos de las trayectorias de cada UAV cada vez que alcanza un waypoint del mission plan 4.3.

4.2. Creación del dataset para el modelo

Una vez generados los datos de trayectorias mediante BB-PLANNER, procedemos a crear el dataset mediante Pandas para nuestro modelo de redes neuronales.

4.2.1. Extracción y procesamiento datos salida BUBBLES BB-PLANNER.

Mediante el script FlightsDataset.py, donde se encuentra la clase FlightsDataset mostrada en el diagrama UML 4.2, cargamos la salida de BB-PLANNER mediante la función *load_raw_bubbles_output*() y extraemos las columnas relevantes de su salida para nuestro modelo. En este caso, las columnas que pueden resultar útiles son las correspondientes con los datos espaciales y temporales de trayectoria, que son: El número de UAV simulado, la categoría del UAV, el número de waypoint, el tiempo, las coordenadas espaciales (x, y, h), los ángulos con respecto a los vectores de las coordenadas (x, y) donde se encuentra el target del siguiente waypoint, la velocidad lineal y angular del UAV, el Rate of Climb o ratio de ascenso del UAV, el tipo de trayectoria del UAV y el tipo de plan de vuelo. A partir de estos datos, generamos un nuevo dataset 4.4 donde vamos a utilizar las diferencias entre una coordenada y la siguiente en el tiempo. En primer lugar, por cada UAV del dataset, convertimos las coordenadas absolutas (x, y, h), a coordenadas relativas, donde la coordenada $(0, 0, 0)$ es el punto de despegue del UAV, y el resto de coordenadas son recalculadas a partir de este nuevo punto inicial. Para la velocidad lineal se sigue el mismo procedimiento, mientras que para los ángulos se calcula su seno y coseno. Una vez hecho esto, por cada UAV, se calcula la diferencia de cada coordenada con respecto a su anterior valor, esto es, $x_t = x_t – x_{i-t}$, $y_t = y_t – y_{t-1}$, $h_t = h_t – h_{t-1}$. Finalmente, se normaliza cada coordenada para que este contenida en el rango $[-1, +1]$, siendo -1 el mínimo valor de la coordenada, y $+1$ el máximo. El proceso de calcular la diferencia entre coordenadas anteriores, hace que la primera coordenada de cada UAV sea nula, ya que no tiene ninguna posición anterior, por lo que la eliminamos del dataset para el proceso de entrenamiento. Esta tarea se lleva a cabo en la función *create_dataset*(). Para



x_coord_diff_norm	,y_coord_diff_norm	,h_coord_diff_norm
-0.99999899525227	,1.0	,-0.9997898372461932
-0.99999899525227	,1.0	,-0.9734655553420648
-0.99999899525227	,1.0	,-0.947812902994825
-0.9999971227678645	,0.999992446284234	,-0.9329314935936479
-1.0	,0.9999860821300051	,-0.93289626611171
-0.9999988125708645	,0.9999895318584653	,-0.9329147605397274
-0.9999973511196213	,0.999992446284234	,-0.9329310532501237
-0.9999953416241614	,0.9999950633196175	,-0.9329491073346169
-0.9999951132724045	,0.9999956581003866	,-0.9329508687087138
-0.999995798327675	,0.9999921488938495	,-0.9329398601206083

Figura 4.4.: Primeros 10 registros del nuevo dataset procesado con las coordenadas centradas en el punto de despegue, las diferencias entre coordenadas en el tiempo y reescaladas en el intervalo $[-1, 1]$ en CSV (se ha omitido la columna que identifica a cada UAV, y la primera coordenada nula).

dataset	100 % dataset		67 % dataset		34 % dataset	
	Nº trayectorias	Nº UAVs	Nº trayectorias	Nº UAVs	Nº trayectorias	Nº UAVs
Train	90058	69	57728	46	28130	23
Validation	29780	23	21500	15	9800	7
Test	29480	23	19209	15	10725	7

Cuadro 4.1.: Información estructura datasets.

separar el dataset en tres datasets distintos para los conjuntos de entrenamiento, validación y test, se hace uso de la función *split_dataset*() que, selecciona aleatoriamente un subconjunto de los UAVs, y asigna todo su registro de coordenadas a uno de los tres conjuntos para el modelo de ML donde corresponda.

Una vez procesados los datos, nuestros datasets estan formados por la información mostrada en la tabla 4.1:

Para utilizar este dataset en nuestro modelo, la clase Trajectory_seq2seq_Dataset lo convierte en un Pytorch.Tensor con el numero de trayectorias a utilizar para la predicción o a predecir que le hayamos indicado. La dimensión de este tensor es $n \times 3$, siendo n el número de trayectorias usadas.

$$\begin{bmatrix} x_t & x_{t+1} & \dots & x_n \\ y_t & y_{t+1} & \dots & y_n \\ h_t & h_{t+1} & \dots & h_n \end{bmatrix} \tag{4.2}$$

Además, la clase Trajectory_seq2seq_Dataset añade una dimensión extra indicando el número de dimensiones y al cargar este objeto en el objeto DataLoader, se añade una dimensión más indicando el dispositivo CUDA en el que se va a ejecutar el modelo.

4.3. Arquitectura del modelo

La arquitectura del modelo corresponde a una arquitectura Seq2Seq [26] (Sequence to Sequence o Secuencia a Secuencia) ya que queremos un modelo que, dada una secuencia de entrada (secuencia de trayectorias pasadas), pueda estimar una secuencia de salida (la secuencia de trayectorias a predecir). Este tipo de arquitecturas es el que usualmente son utilizadas para procesamiento de lenguaje natural[1], en modelos como los de chatGPT. Existen múltiples maneras de implementar esta clase de modelos, desde redes neuronales clásicas como el perceptrón, hasta arquitecturas más sofisticadas con mecanismos de atención. En nuestro caso, vamos a utilizar una arquitectura Seq2Seq mediante un Encoder-Decoder con capas LSTM[2]. Para cada elemento de la secuencia de entrada, cada capa LSTM calcula la siguiente función [27]:

$$
\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}ht{-}1 + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}
$$

donde h_t es el estado oculto en el tiempo t, c_t es el estado de la celda en el tiempo t, x_t es la entrada en el tiempo t, h_{t-1} es el estado oculto de la capa en el tiempo $t-1$ o el estado oculto inicial en el tiempo 0, y i_t, f_t, g_t, o_t son las puertas de entrada, olvido, celda y salida, respectivamente. σ es la función sigmoide, y $\odot$ es el producto de Hadamard. En una LSTM multicapa, la entrada $x_t^{(l)}$ de la capa l-ésima ($l \geq 2$) es el estado oculto $h_t^{(l-1)}$ de la capa anterior multiplicado por el dropout $\delta_t^{(l-1)}$ donde cada $\delta_t^{(l-1)}$ es una variable aleatoria Bernoulli que es 0 con probabilidad = dropout.

El Encoder se encargará de codificar los datos de entrada (trayectorias) para convertirlos en una representación en un espacio latente que el Encoder crea en su salida. A partir de esta representación, el Decoder la recibirá como entrada para decodificarla en la secuencia de salida que será la secuencia de trayectorias siguientes en el tiempo. En primer lugar, haciendo uso de los atributos de la clase TrajectoryModel se deciden los parámetros para el entrenamiento del modelo, los parámetros que utilizaremos son los siguientes:

- dataset_pc: Es el porcentaje del dataset a utilizar durante el entrenamiento, dado que disponemos de un dataset con un cierto tamaño y recursos limitados, este atributo se utiliza para hacer pruebas con un tamaño de dataset menor que el original, agilizando la tarea de entrenamiento del modelo.

- vars_to_predict: Es el nombre de las variables a predecir, se utiliza para decidir qué dimensiones de entrada y salida queremos que estime el modelo, por ejemplo, si queremos entrenar al modelo sobre una única variable espacial de trayectoria para analizar

[1] NLP (Natural Languaje Processing): (`pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html`)

[2] torch.nn.LSTM: (`pytorch.org/docs/stable/generated/torch.nn.LSTM.html#torch.nn.LSTM`)

su rendimiento, o si queremos hacerlo sobre las tres variables espaciales de trayectoria (x, y, h).

- batch_size: Número de muestras que se propagan a través de la red neuronal.
- seq_len: Tamaño de la secuencia de entrada de la muestra. Es el número de coordenadas anteriores que se dan de entrada a la red neuronal para generar la salida.
- num_epochs: Es el número de veces que el dataset de entrenamiento entero es propagado a través de la red neuronal durante el entrenamiento.
- learning_rate: En ML, la tasa de aprendizaje es un parámetro de ajuste en un algoritmo de optimización que determina el tamaño del paso en cada iteración mientras se mueve hacia un mínimo de una función de loss.
- input_size: Es el número de dimensiones usadas en vars_to_predict.
- num_layers: Es el número de capas ocultas del Encoder y del Decoder del modelo Seq2Seq.
- hidden_size: Es el número de neuronas de cada capa oculta del Encoder y del Decoder del modelo Seq2Seq.
- output_len: Tamaño de la secuencia de salida de la muestra. Es el número de coordenadas estimadas posteriores que genera como salida la red neuronal. En nuestro caso, output_len = seq_len

A continuación, se utilizan los datasets de entrenamiento y validación generados en la clase FlightsDataset y se cargan en un objeto creado por la clase Trajectory_seq2seq_Dataset. Esta clase, establece el número de trayectorias anteriores utilizadas para predecir y el número de trayectorias estimadas de salida mediante el atributo seq_len. En nuestro caso, se hicieron pruebas estimando para seq_len = 1, 3, 5 y 10. Este objeto Trajectory_seq2seq_Dataset, se utiliza para crear el objeto Pytorch.DataLoader donde podemos establecer el batch_size para el entrenamiento y el número de hilos de ejecución para cargar los datos en memoria. Seguidamente, se crea el objeto Seq2Seq, con el output_length correspondiente, que contiene al Encoder y al Decoder. Esta clase tiene una función forward() 4.5 que indica al modelo como pasar la información a través del Encoder y el Decoder.

Las clases Encoder y Decoder contienen los atributos necesarios para definirlas, así como su propia función forward() 4.6.

Como función de activación elegimos la función Lineal[3], ya que ofrece un buen rendimiento para tareas de regresión. Se hicieron algunas pruebas con la activación ReLu[4] y LeakyReLu[5], pero no se observó una diferencia significativa de mejora. Como función loss, elegimos MSE[6], ya que ha sido ampliamente usada en la bibliografía de este trabajo y de otros problemas de regresión, además de ser la métrica más versátil para esta clase de problemas. Como método

[3] torch.nn.Linear: (`pytorch.org/docs/stable/generated/torch.nn.Linear.html#torch.nn.Linear`)
[4] torch.nn.ReLU: (`pytorch.org/docs/stable/generated/torch.nn.ReLU.html#torch.nn.ReLU`)
[5] torch.nn.LeakyReLU: (`pytorch.org/docs/stable/generated/torch.nn.LeakyReLU.html#torch.nn.LeakyReLU`)
[6] torch.nn.MSELoss: (`pytorch.org/docs/stable/generated/torch.nn.MSELoss.html`)

```python
class Seq2Seq(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_dim, output_length, device):
        super().__init__()
        self.output_length = output_length
        self.encoder = Encoder(input_size, hidden_size, num_layers).to(device)
        self.decoder = Decoder(input_size, hidden_size, num_layers, output_dim).to(device)

    def forward(self, x, y, teacher_force_ratio):
        hidden, cell = self.encoder(x)
        predictions = []
        dec_input = x[:, -1, :].unsqueeze(1)
        for i in range(self.output_length):
            prev_x, prev_hidden, prev_cell = self.decoder(dec_input, hidden, cell)
            hidden, cell = prev_hidden, prev_cell
            teacher_force = torch.rand(1).item() < teacher_force_ratio
            dec_input = y[:, i, :].unsqueeze(1) if teacher_force else prev_x
            predictions.append(prev_x.squeeze(1))
        outputs = torch.stack(predictions)
        return outputs.permute(1, 0, 2)
```

Figura 4.5.: Clase Seq2Seq, mostrando la función forward().

```python
class Encoder(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers):
        super().__init__()
        self.rnn1 = nn.LSTM(
            input_size  = input_size,
            hidden_size = hidden_size,
            num_layers  = num_layers,
            batch_first = True,
            dropout     = 0.1
        )

    def forward(self, x):
        x, (hidden, cell) = self.rnn1(x)
        return hidden, cell
```

```python
class Decoder(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, output_dim):
        super().__init__()
        self.rnn1 = nn.LSTM(
            input_size  = input_size,
            hidden_size = hidden_size,
            num_layers  = num_layers,
            batch_first = True,
            dropout     = 0.1
        )

        self.output_layer = nn.Linear(hidden_size, output_dim)

    def forward(self, x, input_hidden, input_cell):
        outputs, (hidden, cell) = self.rnn1(x, (input_hidden, input_cell))
        prediction = self.output_layer(outputs)
        return prediction, hidden, cell
```

Figura 4.6.: Clases Encoder y Decoder, mostrando su función forward().

de optimización estocástica, elegimos el algoritmo Adam[7], otro estándar para los problemas de regresión y redes neuronales.

4.3.1. Entrenamiento y validación del modelo de ML

En este momento, pasamos a entrenar el modelo mediante la función train() 4.7 del objeto Seq2Seq.

Para el testeo del modelo, recorremos el dataset de test e introducimos como entrada del modelo una secuencia con el número de coordenadas que queremos estimar (que son las trayectorias pasadas), y calculamos su MSE con la salida que nos proporciona el modelo, que es otro tensor, con el número de estimaciones que hemos introducido en la entrada para las variables de la trayectoria de la entrada, este MSE se divide entre el número de coordenadas a estimar para obtener el MSE medio por coordenada. Este proceso se repite para cada registro del dataset, esto es, suponiendo que escogemos un seq_len = 3 y queremos estimar las coordenadas espaciales (x, y, h), la entrada para el modelo sería:

$$\begin{bmatrix} x_t & x_{t-1} & x_{t-2} \\ y_t & y_{t-1} & y_{t-2} \\ h_t & h_{t-1} & h_{t-2} \end{bmatrix} \tag{4.3}$$

La salida del modelo nos devolvería el siguiente tensor:

$$\begin{bmatrix} x_{t+1} & x_{t+2} & x_{t+3} \\ y_{t+1} & y_{t+2} & y_{t+3} \\ h_{t+1} & h_{t+2} & h_{t+3} \end{bmatrix} \tag{4.4}$$

Para optimizar el modelo, realizamos varias pruebas variando algunos de sus parámetros. A continuación se muestran los resultados del MSE en las tablas 4.2 y 4.3 en el conjunto de entrenamiento, validación y test con las distintas configuraciones. Las pruebas se realizaron con el 34 % y el 67 % del dataset respectivamente.

Los resultados mostrados son con la coordenada x de la trayectoria, pero se realizaron pruebas con las dos coordenadas restantes y y h y se obtuvieron resultados similares tanto de MSE como de tiempo.

Analizando los modelos donde estimamos las coordenadas (x, y, h), se puede observar en la tabla 4.3 que alcanzamos los menores valores de MSE en el conjunto de test cuando estimamos en el corto plazo con menores secuencias temporales. Esta es una tendencia que se repite en los modelos entrenados con una única coordenada como se observa en la tabla 4.2

[7] torch.optim.Adam: (`pytorch.org/docs/stable/generated/torch.optim.Adam.html`)

```python
def train(model, num_epochs, tr_dl, va_dl, loss_fn, optimizer, device):
    loss_hist_train = [0] * num_epochs
    loss_hist_valid = [0] * num_epochs

    model.to(device)
    start_time = time()
    for epoch in range(num_epochs):
        model.train()
        teacher_forcing_ratio = scheduled_sampling(epoch, num_epochs)
        avg_loss = 0
        for i, (history, target) in enumerate(tr_dl):
            seq_input  = history.to(device)
            seq_output = target.to(device)
            # forward
            outputs = model(seq_input, seq_output, teacher_forcing_ratio)
            loss = loss_fn(outputs, seq_output)
            # backward
            optimizer.zero_grad()
            loss.backward()
            # update
            optimizer.step()
            # log
            avg_loss += loss.item() / len(tr_dl)
        model.eval()
        avg_val_loss = 0
        with torch.no_grad():
            for history, target in va_dl:
                outputs = model(history.to(device), seq_output, 0)
                loss = loss_fn(outputs, target.to(device))
                avg_val_loss += loss.item() / len(va_dl)

        elapsed_time = time() - start_time
        loss_hist_train = avg_loss
        loss_hist_valid = avg_val_loss

        print(f"Epoch {epoch :02d}/{num_epochs :02d} \t t={elapsed_time:.0f}s \t"
                f"train_loss={avg_loss:.4f} \t"
                f"valid_loss={avg_val_loss:.4f}"
                )
    return loss_hist_train, loss_hist_valid



def scheduled_sampling(epoch, max_epochs, start_ratio=1.0, end_ratio=0.0, decay_every=5):
    return start_ratio - (start_ratio - end_ratio) * (epoch / max_epochs)
```

Figura 4.7.: Función train().

Variables	seq_len	num_ep	num_lay	hid_sze	Train MSE	Val MSE	Test MSE	Train&Val time (s)
$[x]$	10	512	3	4	0.0049	0.0066	0.0441	13872s
$[x]$	10	256	2	2	0.0095	0.0093	0.0406	13485s
$[x]$	3	256	2	2	0.0127	0.0066	0.0140	13573s
$[x]$	3	256	4	8	0.0035	0.0026	0.0014	13837s
$[x]$	3	256	8	16	0.0029	0.0025	0.0012	13906s
$[x]$	1	256	2	4	0.0104	0.0050	0.0043	13751s

Cuadro 4.2.: Resultados MSE train/val/test con distintos parámetros con el 34 % del dataset y una única variable espacial de coordenada.

Variables	seq_len	num_ep	num_lay	hid_sze	Train MSE	Val MSE	Test MSE	Train&Val time (s)
$[x, y, z]$	3	256	32	16	0.4230	0.4459	1.3000	14218s
$[x, y, z]$	3	256	3	3	0.1224	0.1844	0.4136	14063s
$[x, y, z]$	3	512	3	3	0.0408	0.0512	0.1058	28301s
$[x, y, z]$	3	1024	3	3	0.0166	0.0109	0.0240	56022s
$[x]$	1	1024	3	3	0.0311	0.0123	0.0085	55931s
$[x, y, z]$	5	1024	3	3	0.0137	0.0116	0.0423	56781s
$[x, y, z]$	10	1024	3	3	0.0156	0.0174	0.1231	57094s
$[x, y, z]$	10	1024	3	4	0.0134	0.0166	0.1073	56876s
$[x, y, z]$	10	1024	8	4	0.0594	0.0875	0.6578	58203s
$[x, y, z]$	3	4096	4	8	0.0030	0.0020	0.0060	252236s

Cuadro 4.3.: Resultados MSE train/val/test con distintos parámetros con el 67 % del dataset.

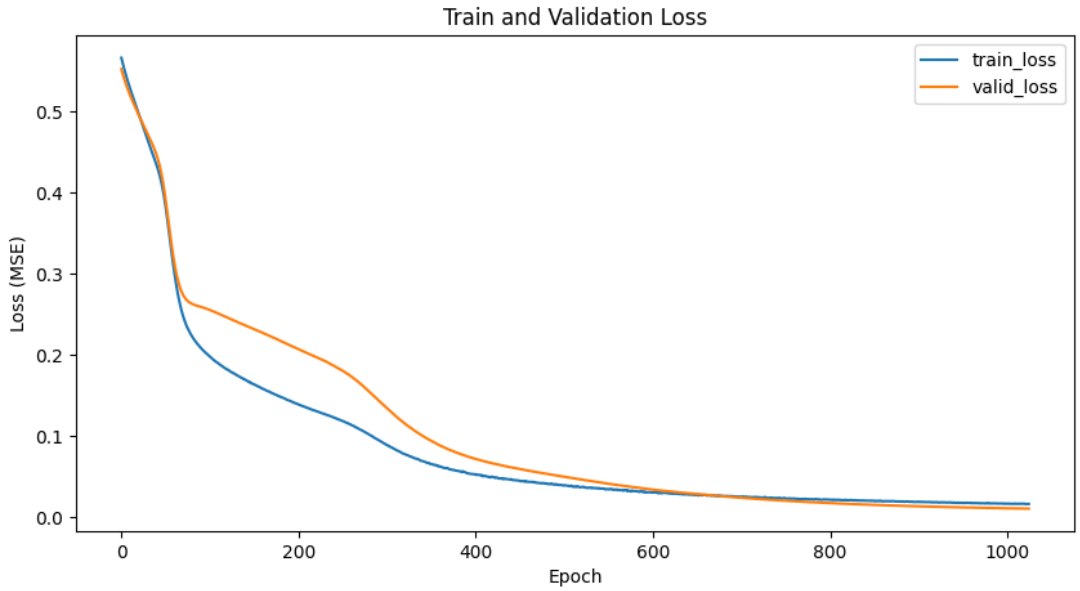


Figura 4.8.: MSE de entrenamiento y validación a lo largo de las epochs de entrenamiento para el modelo estimador de las coordenadas (x, y, h) con menor MSE en el conjunto de test.

En la imagen 4.8 observamos como el modelo empieza a converger en el MSE de entrenamiento y test a partir de las 600 epochs, aunque sigue aprendiendo de los datos a medida que continúan subiendo las epochs. Conseguimos un MSE en el conjunto de entrenamiento de 0.0166 y un MSE en el conjunto de test de 0.024. El principal cuello de botella es el tiempo de entrenamiento de los modelos, como se puede observar, la mayoría de modelos tardaron varias horas en entrenarse, algunos más complejos incluso varios días.

También probamos a entrenar el modelo en el resto de las variables de la trayectoria de las que disponíamos, las diferencias temporales entre (x, y, h) como en el resto de pruebas y las diferencias temporales entre sin y cos de (x, y), se probaron a estimar las 10 siguientes dimensiones de trayectorias partiendo de las 10 anteriores, con 1024 epochs, 2 capas ocultas y 2 neuronas en cada capa oculta. El resultado fue un MSE de entrenamiento de 0.0286, un MSE de validación de 0.0463 y un MSE de test de 0.3324, con un tiempo de entrenamiento de 57566s o 16h aproximadamente. El tiempo medio de inferencia por predicción fue de 0.032s, el resto de modelos con únicamente las coordenadas (x, y, h) tuvieron tiempos similares, excepto cuando se estimaba menos pasos en el tiempo, que llegaba a bajar un orden de magnitud. en la imagen 4.9 se muestra un ejemplo gráfico de la trayectoria simulada por BB-PLANNER y la estimada por nuestro modelo

Como se puede observar por los resultados, el modelo es capaz de alcanzar unos valores de MSE en el conjunto de test suficientes para otorgar trayectorias muy cercanas a las simuladas por BB-PLANNER en las estimaciones a corto plazo (1-5 pasos en el tiempo) y con una única variable de trayectoria en pasos de tiempo más largos (hasta 10 en nuestras pruebas). en la tabla 4.4 se muestra un ejemplo de secuencia de entrada y salida del modelo. Se observa

Tiempo	x_sim	x_est	y_sim	y_est	h_sim	h_est
$t+1$	-0.9981	-0.9931	-0.9981	-0.9942	-0.9905	-0.9928
$t+2$	-0.9991	-0.9929	-0.9991	-0.9949	-0.9955	-0.9990
$t+3$	-0.9997	-0.9954	-0.9997	-0.9819	-0.9987	-0.9897

Cuadro 4.4.: Ejemplo de estimación de trayectoria de (x, y, h) en 3 pasos en el tiempo. Las columnas: _sim es la trayectoria simulada por BB-PLANNER, _est es la trayectoria estimada por nuestro modelo.

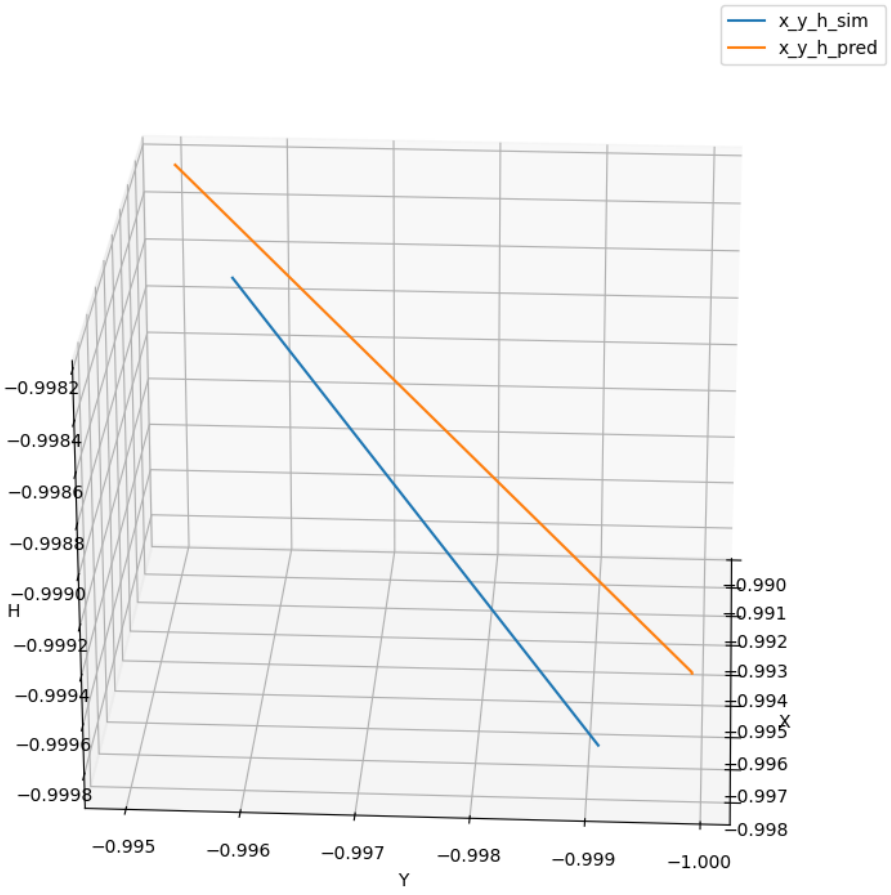


Figura 4.9.: Ejemplo de una de las trayectorias simuladas y su estimación por nuestro modelo.

también que los mayores cuellos de botella en cuanto a tiempo de entrenamiento son el num_epochs, el tamaño del dataset y la longitud de la secuencia a estimar.

5. Conclusión

En este trabajo, hemos realizado una investigación sobre el estado del arte de modelos de predicción de trayectoria en UAVs basados en ML, además de haber implementado, entrenado y testeado un modelo Seq2Seq LSTM para la estimación de trayectoria en un dataset generado por el software BUBBLES BB-PLANNER.
Los resultados del modelo conseguido son similares en cuanto a MSE se refiere a los consultados de la bibliografía, además un análisis gráfico indica que las trayectorias generadas por el modelo concuerdan con las trayectorias simuladas utilizadas para el testeo del mismo, por lo que podemos concluir que el ML es una alternativa viable a los algoritmos clásicos para la predicción de trayectorias en vehículos aéreos no tripulados. Ésta es la misma conclusión a la que han llegado numerosos autores dedicados a distintas áreas donde se hace necesaria una herramienta para la estimación de trayectoria, por ello, desde hace años existe una tendencia a utilizar este tipo de soluciones a tareas como el control aéreo efectivo, la autonomía de pilotaje de vuelo o la simulación de escenarios de vuelo.

Como posibles trabajos futuros, se podrían explorar alternativas a la arquitectura utilizada, desde modelos con mecanismos de atención como los utilizados para las tareas de NLP (Natural Language Processing), hasta combinaciones de distintos modelos que sean capaces de gestionar el resto de datos de los que un drone puede disponer y no únicamente sus coordenadas inerciales. Algunos de estos datos podrían ser distintos parámetros como la velocidad y dirección del viento o las imágenes capturadas por la cámara del drone, que podrían ser utilizadas para compararlas con imágenes de archivo reales del mismo terreno con el fin de determinar con mayor precisión la posición real del UAV.
Otra posible mejora podría ser la simulación de errores en las trayectorias (como errores gaussianos en los datos de trayectoria debido a una mala lectura o, directamente, la eliminación de algunos de los datos de trayectoria simulando una pérdida de la señal) para analizar como se comportaría el modelo frente a ellos, e intentar desarrollar una estrategia para su mitigación.
Finalmente como posible trabajo futuro, hoy en día existe la posibilidad de implementar esta clase de modelos de redes neuronales directamente en hardware en lugar de utilizar procesadores de propósito más general como las CPUs o GPUs. Estas implementaciones se conocen como Edge AI, y aprovechan las ventajas de una implementación directa en el hardware como la velocidad de inferencia para sistemas en tiempo real, el bajo consumo para dispositivos sin fuente de alimentación directa, la eliminación de errores de transmisión de los datos a través de una red inalámbrica o la privacidad de los datos de trayectoria ya que el cómputo se lleva a cabo a nivel local dentro del sistema UAV. Como ejemplo de hardware de Edge AI podemos encontrar a NVIDIA Jetson[1], entre otros. En caso de necesitar una solución más personalizada y con mayor control, se podría utilizar un lenguaje de descripción de hardware

[1]NVIDIA JETSON: (`www.nvidia.com/es-es/autonomous-machines/embedded-systems/`)

como Verilog[2][3][4] o VHDL para diseñar el chip desde 0[5][6].

[2]chipon (`github.com/rohittp0/chipon`)
[3]nngen (`github.com/NNgen/nngen`)
[4]hls4ml (`github.com/fastmachinelearning/hls4ml`)
[5]qkeras (`github.com/google/qkeras/`)
[6]aba-blog (`aba-blog.xyz/dnn-to-chip-1/index.html`)

A. Relación con los ODS de la agenda 2030

Relación con los ODS de la agenda 2030 [1]

ODS 2. En concreto la meta **2.3**, ya que sistemas de drones autónomos pueden ayudar a la gestión, análisis y seguridad de las tierras agrícolas, como por ejemplo en [28, 29, 30]. Y, como se ha visto en este trabajo, uno de los componentes necesarios para desarrollar la autonomía de vuelo en UAVs es la estimación de la trayectoria:
2.3. Para 2030, **duplicar la productividad agrícola y los ingresos de los productores de alimentos en pequeña escala**, en particular las mujeres, los pueblos indígenas, los agricultores familiares, los pastores y los pescadores, entre otras cosas **mediante un acceso seguro y equitativo a las tierras**, a otros recursos de producción e insumos, conocimientos, servicios financieros, mercados **y oportunidades para la generación de valor añadido y empleos no agrícolas**.

ODS 3. En concreto la meta **3.6**, para la monitorización y seguridad de carreteras [2] [31], y el uso de UAVs para transporte de medicamentos [3] [32]:
3.6 Para 2020, reducir a la mitad el número de muertes y lesiones causadas por accidentes de tráfico en el mundo.

ODS 11. En concreto las metas **11.1**, con el aumento del acceso a servicios como el transporte y despliegue de medicamentos, **11.2**, **11.5**, con la monitorización de desastres naturales. **11.6**, con la monitorización de la calidad del aire. **11.b** [4] [33, 34, 35, 36, 37]:
11.1 De aquí a 2030, asegurar el **acceso de todas las personas a** viviendas y **servicios básicos adecuados, seguros y asequibles** y mejorar los barrios marginales.
11.2 De aquí a 2030, **proporcionar acceso a sistemas de transporte seguros, asequibles, accesibles y sostenibles para todos y mejorar la seguridad vial**, en particular mediante la ampliación del transporte público, prestando especial atención a las necesidades de las personas en situación de vulnerabilidad, las mujeres, los niños, las personas con discapacidad y las personas de edad.
11.5 De aquí a 2030, **reducir significativamente el número de muertes causadas por los desastres, incluidos los relacionados con el agua, y de personas afectadas por ellos, y reducir considerablemente las pérdidas económicas directas provocadas por los desastres** en comparación con el producto interno bruto mundial, haciendo especial

[1] Relación con los ODS de la agenda 2030 (`www.un.org/sustainabledevelopment/es/`)
[2] highways.gov: (`highways.dot.gov/media/4556`)
[3] European Pharmaceuticals Review (`www.europeanpharmaceuticalreview.com/article/103799/medicine-delivery-drone-safety-and-quality/`)
[4] EUROCONTROL (`www.eurocontrol.int/article/revolutionising-mobility-cities-through-uavs-airspace-design-`

Objetivos de desarrollo sostenible	Alto	Medio	Bajo	No procede
ODS 1. Fin de la pobreza.				X
ODS 2. Hambre 0.			X	
ODS 3. Salud y bienestar.			X	
ODS 4. Educación de calidad.				X
ODS 5. Igualdad de género.				X
ODS 6. Agua limpia y saneamiento.				X
ODS 7. Energía asequible y no contaminante.				X
ODS 8. Trabajo decente y crecimiento económico.				X
ODS 9. Industria, innovación e infraestructuras.				X
ODS 10. Reducción de las desigualdades.				X
ODS 11. Ciudades y comunidades sostenibles.		X		
ODS 12. Producción y consumo responsables.				X
ODS 13. Acción por el clima.				X
ODS 14. Vida submarina.				X
ODS 15. Vida de ecosistemas terrestres.				X
ODS 16. Paz, justicia e instituciones públicas.				X
ODS 17. Alianzas para lograr objetivos.				X

Cuadro A.1.: Relación del presente TFM con los ODS de la agenda 2030

hincapié en la protección de los pobres y las personas en situaciones de vulnerabilidad.
11.b De aquí a 2020, aumentar considerablemente el número de ciudades y asentamientos humanos que adoptan e implementan políticas y planes integrados para promover la inclusión, **el uso eficiente de los recursos, la mitigación del cambio climático y la adaptación a él y la resiliencia ante los desastres**, y desarrollar y poner en práctica, en consonancia con el Marco de Sendai para la Reducción del Riesgo de Desastres 2015-2030, **la gestión integral de los riesgos de desastre a todos los niveles**.

B. Bibliografía

[1] SESAR Joint Undertaking. *U-space ConOps and architecture (edition 4).* CORUS-XUAM, EUROCONTROL. SESAR Joint Undertaking, 2023.

[2] Andreas Zell. *Simulation neuronaler Netze [Simulation of Neural Networks] (in German) (1st ed.).* Addison-Wesley, 2003.

[3] Hinton G. & Williams R. Rumelhart, D. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[4] Chuquitarco Jiménez C. A. Vico Navarro J. & Balbastre Tejedor J. V. Claramunt Puchol, C. U-space separation management service: concept definition and validation., 2022. SESAR Innovation Days (SIDS), Budapest, Hungary (2022, diciembre 14).

[5] Vera-Vélez N. & Balbastre-Tejedor J. V. Claramunt-Puchol, C. Concept of separation management for uas in the u-space. *Zenodo*, 2022.

[6] Claramunt-Puchol C. & Chuquitarco-Jimenéz C. Balbastre-Tejedor, J. V. Availability note for bubbles mock-up. *Zenodo*, 2022.

[7] Ramon Dalmau-Codina and Eric Allard. Air traffic control using message passing neural networks and multi-agent reinforcement learning. In *Air Traffic Control Using Message Passing Neural Networks and Multi-Agent Reinforcement Learning*, 09 2020.

[8] Duc-Thinh Pham. *Machine learning-based flight trajectories prediction and air traffic conflict resolution advisory.* Theses, Université Paris sciences et lettres, June 2019.

[9] Hong-Cheol Choi, Chuhao Deng, and Inseok Hwang. Hybrid machine learning and estimation-based flight trajectory prediction in terminal airspace. *IEEE Access*, 9:151186–151197, 2021.

[10] Min Xue. *UAV Trajectory Modeling Using Neural Networks.* 2017.

[11] Zhengyi Wang, Man Liang, and Daniel Delahaye. Short-term 4D Trajectory Prediction Using Machine Learning Methods. In *SID 2017, 7th SESAR Innovation Days*, Belgrade, Serbia, November 2017.

[12] Peng Shu, Chengbin Chen, Baihe Chen, Kaixiong Su, Sifan Chen, Hairong Liu, and Fuchun Huang. Trajectory prediction of uav based on lstm. In *2021 2nd International Conference on Big Data & Artificial Intelligence & Software Engineering (ICBASE)*, pages 448–451, 2021.

[13] Gang Zhong, Honghai Zhang, Jiangying Zhou, Jinlun Zhou, and Hao Liu. Short-term 4d trajectory prediction for uav based on spatio-temporal trajectory clustering. *IEEE Access*, 10:93362–93380, 2022.

[14] You Wu, Hongyi Yu, Jianping Du, Bo Liu, and Wanting Yu. An aircraft trajectory prediction method based on trajectory clustering and a spatiotemporal feature network. *Electronics*, 11(21), 2022.

[15] Nora Schimpf, Zhe Wang, Summer Li, Eric J. Knoblock, Hongxiang Li, and Rafael D. Apaza. A generalized approach to aircraft trajectory prediction via supervised deep learning. *IEEE Access*, 11:116183–116195, 2023.

[16] Sifan Chen, Baihe Chen, Peng Shu, Zhensheng Wang, and Chengbin Chen. Real-time unmanned aerial vehicle flight path prediction using a bi-directional long short-term memory network with error compensation. *Journal of Computational Design and Engineering*, 10(1):16–35, 11 2022.

[17] Stefan Becker., Ronny Hug., Wolfgang Huebner., Michael Arens., and Brendan T. Morris. Generating synthetic training data for deep learning-based uav trajectory prediction. In *Proceedings of the 2nd International Conference on Robotics, Computer Vision and Intelligent Systems - ROBOVIS*, pages 13–21. INSTICC, SciTePress, 2021.

[18] Yunhong Yang, Xingzhong Xiong, and Yuehao Yan. Uav formation trajectory planning algorithms: A review. *Drones*, 7(1), 2023.

[19] Manuel Mateos Villar. Machine learning for aircraft trajectory prediction: a solution for pre-tactical air traffic flow management. In *Machine learning for aircraft trajectory prediction: a solution for pre-tactical air traffic flow management*, 2022.

[20] Carole G. Prevost, Andre Desbiens, and Eric Gagnon. Extended kalman filter for state estimation and trajectory prediction of a moving object detected by an unmanned aerial vehicle. In *2007 American Control Conference*, pages 1805–1810, 2007.

[21] Guotao Xie, Hongbo Gao, Lijun Qian, Bin Huang, Keqiang Li, and Jianqiang Wang. Vehicle trajectory prediction by integrating physics- and maneuver-based approaches using interactive multiple models. *IEEE Transactions on Industrial Electronics*, 65(7):5999–6008, 2018.

[22] Adam Houenou, Philippe Bonnifait, Véronique Cherfaoui, and Wen Yao. Vehicle trajectory prediction based on motion model and maneuver recognition. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4363–4369, 2013.

[23] Amal Mahmoud and Ammar Mohammed. *A Survey on Deep Learning for Time-Series Forecasting*, pages 365–392. Springer International Publishing, Cham, 2021.

[24] Jui-Chan Huang, Kuo-Min Ko, Ming-Hung Shu, and Bi-Min Hsu. Application and comparison of several machine learning algorithms and their integration models in regression problems. *Neural Computing and Applications*, 32(10):5461–5469, May 2020.

[25] Hetal Bhavsar and Amit Ganatra. Application and comparison of several machine learning algorithms and their integration models in regression problems. *International Journal of Soft Computing and Engineering (IJSCE)*, 2:74–81, 2012.

[26] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks, 2014.

[27] Haşim Sak, Andrew Senior, and Françoise Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition, 2014.

[28] Jeongeun Kim, Seungwon Kim, Chanyoung Ju, and Hyoung Il Son. Unmanned aerial vehicles in agriculture: A review of perspective of platform, control, and applications. *IEEE Access*, 7:105100–105115, 2019.

[29] G. Grenzdörffer and Bernd Teichert. The photogrammetric potential of low-cost uavs in forestry and agriculture. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci.*, XXXVII, 01 2008.

[30] J. Gago, C. Douthe, R.E. Coopman, P.P. Gallego, M. Ribas-Carbo, J. Flexas, J. Escalona, and H. Medrano. Uavs challenge to assess water stress for sustainable agriculture. *Agricultural Water Management*, 153:9–19, 2015.

[31] Fatma Outay, Hanan Abdullah Mengash, and Muhammad Adnan. Applications of unmanned aerial vehicle (uav) in road safety, traffic and highway infrastructure management: Recent advances and challenges. *Transportation Research Part A: Policy and Practice*, 141:116–129, 2020.

[32] Katy Surman and David Lockey. Unmanned aerial vehicles and pre-hospital emergency medicine. *Scandinavian Journal of Trauma, Resuscitation and Emergency Medicine*, 32(1):9, Jan 2024.

[33] Farhan Mohammed, Ahmed Idries, Nader Mohamed, Jameela Al-Jaroodi, and Imad Jawhar. Uavs for smart cities: Opportunities and challenges. In *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 267–273, 2014.

[34] Nader Mohamed, Jameela Al-Jaroodi, Imad Jawhar, Ahmed Idries, and Farhan Mohammed. Unmanned aerial vehicles applications in future smart cities. *Technological Forecasting and Social Change*, 153:119293, 2020.

[35] H. Une and T. Nakano. Role of geospatial information for disaster risk management as exemplified in recent large earthquakes in japan. *Advances in Cartography and GIScience of the ICA*, 1:21, 2019.

[36] A. Román, A. Tovar-Sánchez, D. Roque-Atienza, I.E. Huertas, I. Caballero, E. Fraile-Nuez, and G. Navarro. Unmanned aerial vehicles (uavs) as a tool for hazard assessment: The 2021 eruption of cumbre vieja volcano, la palma island (spain). *Science of The Total Environment*, 843:157092, 2022.

[37] P. Ezquerro, G. Bru, I. Galindo, O. Monserrat, J.C. García-Davalillo, N. Sánchez, I. Montoya, R. Palamà, R.M. Mateos, R. Pérez-López, E. González-Alonso, R. Grandin, C. Guardiola-Albert, J. López-Vinielles, J.A. Fernández-Merodo, G. Herrera, and M. Béjar-Pizarro. Analysis of sar-derived products to support emergency management during volcanic crisis: La palma case study. *Remote Sensing of Environment*, 295:113668, 2023.