UNIVERSITAT POLITÈCNICA DE VALÈNCIA

School of Informatics

Tracking of UAVs using Kalman Filters and Deep Learning

End of Degree Project

Bachelor's Degree in Informatics Engineering

AUTHOR: Santos Escriche, Eduardo

Tutor: Onaindia de la Rivaherrera, Eva

ACADEMIC YEAR: 2021/2022

Acknowledgements

Firstly, I would like to thank Dr. Eva Onaindía de la Rivaherrera for proposing this topic to me, as well as for her help, guidance and support throughout the entire development of the project.

We would also like to thank Ndoni Adelica, the contact person of the NATO Communications and Information Agency (NCIA), for permitting us to use the dataset of the competition "Drone identification and tracking", part of the International Conference on Military Communication and Information Systems (ICMCIS), which was published on Kaggle at https://www.kaggle.com/c/icmcis-drone-tracking/.

Moreover, this project has been supported by the *Beca de Colaboración* awarded to us by the *Ministerio de Educación y Formación Profesional* for a project titled "Aprendizaje de modelos de comportamiento de Vehículos Aéreos no Tripulados (UAV)".

Resum

Els vehicles aeris no tripulats (UAVs) estan cada vegada més disponibles i són més capaços, utilitzant-se per a una gran varietat de propòsits, com la vigilància, la supervisió d'entorns perillosos, el lliurament de càrrega, l'enregistrament, la cerca i rescat, i molts més. El seguiment dels vehicles aeris no tripulats s'ha convertit en una tasca crucial per a fer front a la creixent utilització d'aquesta mena de vehicles per a plans maliciosos i delictius, així com per a amenaces i vulnerabilitats de seguretat. En aquest treball, presentem dos enfocaments diferents per al seguiment d'UAV, un basat en els àmpliament utilitzats Filtres de Kalman (KF) per a extraure informació de mostres sorolloses i l'altre basat en algorismes d'Aprenentatge Profund. El nostre objectiu és comparar el rendiment d'aquestes dues tècniques contraposades, el model de caixa blanca de KF que empra les equacions del model dinàmic de l'UAV i un model de caixa negra implementat amb xarxes neuronals. Per a l'avaluació experimental, utilitzem un dataset proporcionat per l'OTAN que conté una combinació de dades de sensors de radar i de sensors de cerca per ràdio direcció. Els resultats d'aqueixa avaluació experimental mostren que tots dos tipus de models considerats són capaços de proporcionar un bon rendiment per a aquest problema específic. A més, es realitza un estudi comparatiu del seu rendiment per a discernir les característiques associades al comportament de cada model.

Paraules clau: Seguiment d'UAVs, Filtres de Kalman, Aprenentatge Profund, Xarxes Neuronals

Resumen

Los vehículos aéreos no tripulados (UAVs) están cada vez más disponibles y son más capaces, utilizándose para una gran variedad de propósitos, como la vigilancia, la supervisión de entornos peligrosos, la entrega de carga, la grabación, la búsqueda y rescate, y muchos más. El seguimiento de los vehículos aéreos no tripulados se ha convertido en una tarea crucial para hacer frente a la creciente utilización de este tipo de vehículos para planes maliciosos y delictivos, así como para amenazas y vulnerabilidades de seguridad. En este trabajo, presentamos dos enfoques diferentes para el seguimiento de UAV, uno basado en los ampliamente utilizados Filtros de Kalman (KF) para extraer información de muestras ruidosas y el otro basado en algoritmos de Aprendizaje Profundo. Nuestro objetivo es comparar el rendimiento de estas dos técnicas contrapuestas, el modelo de caja blanca de KF que emplea las ecuaciones del modelo dinámico del UAV y un modelo de caja negra implementado con redes neuronales. Para la evaluación experimental, utilizamos un dataset proporcionado por la OTAN que contiene una combinación de datos de sensores de radar y de sensores de búsqueda por radio dirección. Los resultados de esa evaluación experimental muestran que ambos tipos de modelos considerados son capaces de proporcionar un buen rendimiento para este problema específico. Además, se realiza un estudio comparativo de su rendimiento para discernir las características asociadas al comportamiento de cada modelo.

Palabras clave: Seguimiento de UAVs, Filtros de Kalman, Aprendizaje Profundo, Redes Neuronales

Abstract

Unmanned Aerial Vehicles (UAVs) have become increasingly available and capable. They are now used for a large variety of purposes, such as surveillance, monitoring dangerous environments, cargo delivery, filming, search and rescue, and many more.

Tracking UAVs has become a crucial task to address the growing utilization of UAVs for malicious and criminal schemes and security threats and vulnerabilities. In this work, we present two different approaches for UAV tracking, one based on the widely used Kalman Filters (KF) for extracting information from noisy samples and the other based on Deep Learning algorithms. We aim to compare the performance of these two contrasting techniques, the white-box model of KF that employs the dynamic model equations of the UAV and a black-box model implemented with Neural Networks. For the experimental evaluation, we used a dataset provided by the NATO agency that contains a mixture of data from radar and radio direction finding sensors. The results from that experimental evaluation show that both of the considered types of models are able to provide good performance for this specific problem. Moreover, a comparative study of their performance is carried out in order to discern the characteristics associated with the behavior of each model.

Key words: UAV tracking, Kalman Filter, Deep Learning, Neural Networks

Contents

List of Figures

List of Tables

CHAPTER 1 Introduction

This chapter serves as an introduction to the work described in the subsequent chapters of this project. In that sense, Section 1.1 describes the main motivations behind the selection of the chosen topic. Then, the main objectives that we will try to fulfill throughout this project are presented in Section 1.2. Finally, Section 1.3 details the structure of this work.

1.1 Motivation

My first introduction to the topic of tracking UAVs was when Dr. Eva Onaindía de la Rivaherrera, professor at the Department of Computer Systems and Computation (DSIC) and my tutor for this project, proposed that I participate in this research field in the context of the *Beca de Colaboración* awarded to us by the *Ministerio de Educación y Formación Profesional* for a project titled "Aprendizaje de modelos de comportamiento de Vehículos Aéreos no Tripulados (UAV)". This is a fellowship intended for university students to get started to carry out research tasks in university departments.

The topic of tracking UAVs was of great appeal to me because of the increasingly predominant presence of UAVs in many different contexts (videography, photography, security, monitoring, etc.), and the associated necessity of being able to reliably track their trajectory in order to guarantee their safety and proper behavior.

Moreover, the large amount of complexities associated with the tracking problem led me to believe that this project would be both interesting and challenging, and that I could learn a lot from its development.

During the initial stages of the project, when we searched for and analyzed the main methods currently being employed to tackle this problem, we soon realized that we could make a distinction between two different groups of approaches: Kalman Filter-based models, and Deep Learning models. There are some striking differences between those two approaches, so the fact that both of them were extensively utilized in the literature motivated us to comparatively study their behavior for a specific dataset, seeking to deduce the main advantages and disadvantages associated with each model.

Furthermore, the contrast between the white-box and explainable nature of the Kalman Filter and the black-box nature of the Deep Learning models also motivated this comparison as a way of determining the characteristics of the trade-off associated with the explainability of a model and its performance.

1.2 Objectives

The main objectives that this project intends to fulfill are the following:

1. Identify the main techniques and approaches related to the task of tracking a UAV.
2. Study and analyze the sensor data contained in a publicly available dataset, and prepare it for its usage by the models.
3. Investigate the theoretical characteristics and the performance of a linear Kalman Filter model for the problem of tracking a UAV.
4. Investigate the theoretical characteristics and the performance of different Deep Learning models for the problem of tracking a UAV.
5. Compare the performances of the considered Kalman Filter and Deep Learning models, and extract conclusions about their behavior in this specific problem.

1.3 Organization of the document

This work is structured as follows.

First, in Chapter 2, we discuss the problem of tracking a UAV as well as the main techniques and approaches that have previously been considered in order to address that problem. In that sense, Section 2.1, details the main characteristics that need to be considered when tackling the problem of tracking a UAV. Then, in Section 2.2 we showcase some of the most relevant works related to this topic and explain the main ideas of their proposed techniques and approaches.

Next, in Chapter 3, we present the data utilized in order to evaluate the performance of the different models discussed in this work. In that sense, Section 3.1 details the contents of the dataset we have considered. Then, in Section 3.2, we focus on describing the parts of that dataset that we decided to select for this project, as well as the preprocessing steps we followed in order to prepare that data for its usage by the different models in consideration. Finally, Section 3.3 details the metrics we have considered in order to evaluate the performance of the models.

The first model considered in this work is the Kalman Filter, which we discuss in Chapter 4. In that chapter, we first provide a theoretical overview of the characteristics and the behavior of this model in Section 4.1. Then, in Section 4.2, we specify the design decisions we took in order to obtain an implementation of this model. Finally, in Section 4.3, we show the results obtained after applying the error metrics to the predictions generated by the Kalman Filter model.

Afterwards, we discuss the different Deep Learning models we have considered in this project in Chapter 5. First, we provide a theoretical overview of the different models in Section 5.1. Later, in Section 5.2, we explain the main design decisions regarding the implementation of those models. Finally, in Section 5.3, we show the errors that resulted from the generated predictions.

Then, a comparison of the performance of the Kalman Filter and Deep Learning models is presented in Chapter 6. First, in Section 6.1, we explore the differences in the errors generated by the predictions of the different models. Later, Section 6.2 contains a detailed study of the computation time required by the different models. Then, in Section 6.3, we provide a final comparison of the main characteristics and performance of the studied models.

Finally, Chapter 7 summarizes the main conclusions that can be derived from the work carried out in this project and specifies some future research directions that could be followed in order to expand on this work.

CHAPTER 2 Related Work

This chapter is structured as follows. Section 2.1 defines the problem of tracking a UAV, as well as the difficulties and benefits associated with developing approaches that address this problem. Later, in Section 2.2, we showcase a series of works that present techniques and approaches aimed at this problem of tracking a UAV.

2.1 Tracking of UAVs

In this section, we will explain the problem of tracking a UAV, the main difficulties associated with it, why it is a problem of great relevance nowadays, and what benefits could be obtained from the development of a reliable and accurate approach.

The problem of tracking a UAV consists in predicting its next state, taking into account its previous states as reported by the data obtained from some sensors. The state of a UAV may be defined, for instance, by its position at a given time.

The complexity of this problem mainly stems from the difficulty of obtaining accurate reports of the position of the UAV from the sensors, which typically introduces a lot of noise in the data report. The noise adds a great deal of difficulty to the process of defining and training an accurate model capable of reliably predicting the position of the UAV. In addition, there are other sources of noise that contribute to the complexity associated with identifying the underlying dynamics of the UAV's flight. Most of these additional sources of noise are related to the weather, for instance, wind, rain, snow, etc.

Despite the difficulty of predicting the trajectory of a UAV, there is a lot of interest in finding approaches that provide high accuracy in the prediction of the position of UAVs during their flight. This interest mainly stems from the technological advances and the proliferation of UAVs, which in turn give rise to security threats, vulnerabilities, as well as their utilization in malicious or criminal schemes. Being able to reliably predict the next position of a UAV during its flight would allow the development of air defense systems that alleviate those potential issues. Other scenarios that would also benefit from the development of such a system are related to the usage of UAVs in the context of sustainable cities or in the context of enhancing the coverage of communication networks. Moreover, there are many other application purposes for UAVs, such as environment monitoring, agriculture, wildlife monitoring, photography, etc.

Before delving into the most common techniques and approaches to the task of tracking a UAV, we will mention some of the main data sources that are currently utilized in order to obtain readings of the movement of UAVs:

- **Radio Frequency (RF) Analyzers:** This type of equipment consists of one or more antennas that receive radio waves and a processor that analyzes the RF spectrum. Its usage aims at detecting radio communication between a UAV and its controller.
- **Acoustic Sensors:** One or more microphones are utilized in order to detect the sound made by a UAV in order to calculate its direction. Adding more sets of microphones enables the approximated triangulation of the position of the UAV.
- **Optical Sensors:** Standard cameras and optical sensors that utilize infrared or thermal imaging are some of the most commonly used equipment to detect UAVs.
- **Radar:** Another very common source of UAV data is radar sensors. This type of sensor works by sending out a signal and receiving its reflection, being able then to measure the direction and position of the UAV.

2.2 Techniques and approaches

In this section, we will describe the main ideas of several works that propose different techniques and approaches related to our problem of tracking UAVs using radar sensor data. In that sense, we found a scarcity of literature that considers exactly the problem of tracking a UAV target. As a consequence, we identified two different categories of related works. The first category reviews proposals related to the problem of tracking, including some approaches that do not specifically consider UAVs but that propose an approach of great similarity to the characteristics considered in this project. The second category walks through some approaches that address different problems related to UAVs, such as detection or classification, which are also related to the problem addressed in this project because of their consideration of different characteristics of UAVs.

2.2.1. Tracking-related approaches

Firstly, we will describe a series of works that propose different techniques and approaches for the problem of tracking a target. We have mainly included works that consider UAVs as the target. However, given that the literature on this particular topic is not especially abundant, we also decided to include some works that consider other types of vehicles as targets while still being related to the problem that concerns us regarding the followed approach.

Neural network-based approaches have been used to model the trajectory of a small UAV without the need to know its control system or identify its aerodynamic parameters [3]. Instead of collecting real trajectory data from flight tests, this work uses a mathematical vehicle model to generate trajectory data that could be utilized for the training and testing phases of the proposed neural network. The data is then used to train the multilayered neural network model, which considers the state of the UAV and wind conditions in order to make predictions. The results obtained when testing this model were considered very promising by the author, since the trained neural network was shown to be able to predict 4D trajectories accurately.

Similarly, in [4], Deep Neural Network (DNN) and Convolutional Neural Network (CNN) models are used in order to estimate the distance between an owned quadrotor UAV and an intruder fixed-wing UAV in order to be able to assess the risk of a mid-air collision. As a previous step to the application of those models, a vision-based object detection method is also developed in order to provide the data to those models. The proposed models are then trained using synthetic images from animation software and then validated considering both synthetic and real flight videos. The obtained results show that the proposed scheme is able to successfully detect and estimate the distance between the UAVs.

Another example of a work that proposes a neural network-based approach is [5], where a stacked Bidirectional and Unidirectional Long Short-Term Memory (SBULSTM) network is proposed in order to predict the position of the UAV during cruise operation. In that sense, the defined model considers the four positions before the current time to predict the position at the next time. The dataset used to evaluate the performance of the proposed model consists of 25 UAV patrol flight paths collected by airborne sensors. Taking into account the obtained results and the analysis carried out by the authors, the proposed model is deemed to provide good performance, obtaining a high prediction accuracy.

There are also other works that approximate the problem of tracking an autonomous vehicle [6]. This latter work presents an LSTM model that allows autonomous vehicles to predict the motion of surrounding vehicles on freeways taking into account their interactions. The datasets considered for the experimental evaluation of the proposed model are publicly available, and they contain real freeway traffic captured at 10 Hz over a period of 45 minutes. Moreover, the proposed model is compared to other vehicle motion prediction approaches. The results show that the proposed model outputs less prediction error compared to other approaches.

Other interesting and somehow related works tackle the problem of efficiently predicting the non-linear and unknown motion of an obstacle in the path of a UAV in order to avoid it [7]. In this sense, the proposal of this work is an efficient method to generate the predictions considering the recent observations, using online training of an LSTM neural network. Those predictions are then utilized to select a velocity that avoids colliding with a given probability. Lastly, a statistical model is utilized in order to verify that the actual motion of the obstacle remains within certain bounds with a given probability.

A different application of UAVs is considered in [8], where the scenario in question consists of using a UAV to improve 5G communication coverage. In that scenario, the UAV is affected by various circumstances (wind, time delay during the air communication, etc.) that result in the base stations producing inaccurate beamforming, which generates an unnecessary capacity loss. This work then proposes a novel Recurrent Neural Network (RNN) based model in order to predict the communication location of the UAV. This approach is then tested in various simulations, and the authors conclude that the proposed RNN-based model is able to generate highly accurate predictions, which can lead to more precise beamforming by the base stations, resulting in a more effective and reliable communication system.

As can be observed in the previously shown works, the current state-of-the-art methods for UAV tracking are mostly neural network-based. However, there also exist other approaches which exploit more traditional and conventional methods of providing reliable tracking, which in turn are more varied in terms of their underlying algorithms as opposed to neural networks. Amongst these techniques, the Kalman Filter is a very popular algorithm for tracking vehicles in noisy environments.

One such example of a more traditional approach to the task of UAV tracking is shown in the proposal of [9], where the Uncertainty and Error-Aware Kalman Filter (UEAKF) model is presented. This non-linear model is aimed at outperforming other existing models in the case of unknown and noisy environments. This work also discusses some of the main disadvantages present in some of those other commonly used models, such as the Kalman Filter (KF), the Unscented Kalman Filter (UKF), the Extended Kalman Filter (EKF), the Particle Filter (PF), or the Diffusion Map Kalman (DMK) model. An analysis is then carried out in order to evaluate the performance of the proposed model and to compare it with that obtained using PF-based and DMK-based models. The results of that analysis show that the proposed UEAKF model is capable of providing a higher accuracy in tracking UAVs compared to the other considered models.

Another work that deals with KF-based models for tracking UAVs is [10], where the EKF, UKF, and Unbiased Converted-Measurement Kalman Filter (UCMKF) are considered. This work then focuses on extending these algorithms in order to consider Earth-Centered Earth-Fixed (ECEF) coordinates instead of East-North-Up (ENU) coordinates, aiming at filtering out the random errors that occur when considering mobile radars, which can cause filter performance degradation when using that coordinate system. A performance evaluation is then carried out in order to compare the behavior of the algorithms using each coordinate system. The results show that the proposed approach of extending the KF-based algorithms when considering ECEF coordinates is able to guarantee the accuracy of the filtering process, being able to track the UAV effectively.

2.2.2. UAV-related approaches

We also identified some works that propose techniques for detecting or classifying UAVs. These problems are related to the problem of tracking because they also consider different characteristics of UAVs, as well as because of the data that is used to tackle such problems.

In that sense, the first work that we can mention is [11], where the problem is about classifying drones utilizing radar signals. In order to approach this problem, the authors propose applying Convolutional Neural Networks (CNNs) to the Short-Time Fourier Transform (STFT) spectrograms of simulated radar signals reflected from the drones. Drones have various characteristics that may impact the spectrograms, such as the blade length and the blade rotation rates. Data is collected under the X-band and the W-band from various radar simulation scenarios and is then utilized in order to evaluate the performance of the proposed model. The results show that the proposed approach is able to provide an F1 score of 0.816 ± 0.011 while remaining robust to the drone blade pitch.

A similar problem is considered in [12], which addresses the scenario of classifying UAV and micro-UAVs and discriminating UAV tracks from non-UAV tracks. The reason why this problem is of great importance is that the proliferation of UAVs has become a threat to air defense systems, especially under different weather conditions such as rain or snow, thus requiring the development of an approach that can alleviate this issue. This work proposes various classification techniques that use kinematic and RF characteristics of the targets and then tests them considering real data. The methods considered for the classification problem are Linear Support Vector Machines, Gaussian Support Vector Machines, Convolutional Neural Networks (CNNs), and Softmax Classifier. Moreover, three different approaches are utilized in order to extract the features considered by those models: user-defined features, Principal Component Analysis (PCA) features, and autoencoder-extracted features. The results of the carried out analysis, using certain combinations of those models and feature extraction approaches, show that the CNN model provides the best performance. Moreover, the autoencoder-extracted features appear to perform better than user-defined and PCA features in the case of the Support Vector Machines methods.

Moreover, [13] tries to address the problem of providing accurate and robust drone detection by proposing a motion-based method called Multi-Scale Space Kinematic detection method (MUSAK). This method is able to harness the motion patterns by extracting 3D, pseudo 3D and 2D kinematic parameters and then considers three Gated Recurrent Units (GRU)-based detection branches for drone recognition. The performance evaluation of the proposed MUSAK model is carried out on a hybrid dataset consisting of public datasets and self-collected data with motion labels. The obtained results and the subsequent analysis show that the model is able to accomplish a higher performance compared with previous state-of-the-art motion-based methods.

CHAPTER 3

UAV tracking dataset

In this chapter, we will first describe the dataset used in this project in Section 3.1. Then, Section 3.2 specifies the parts of that dataset that we decided to select for this project, as well as the preprocessing steps carried out to prepare the data for the experiments described in the following chapters. Finally, in Section 3.3, we provide an overview of the problem of tracking a UAV and an indication of the metrics we will use to measure the performance of the proposed models.

3.1 NCIA Dataset

The dataset selected for this project was made publicly available in Kaggle (a crowd-sourced platform where data scientists can train and solve challenges) [1] by the NATO Communications and Information Agency (NCIA) for the International Conference on Military Communication and Information Systems (ICMCIS) drone detection challenge [14]. Throughout this project, we will refer to this dataset as the NCIA dataset since that is the organization that provided the data.

The ICMCIS drone detection challenge arises from the interest of NATO in developing capabilities that can help protect people and equipment against the threat of the misuse of UAVs. In particular, the goal of this challenge is to detect, classify and track different UAVs as they fly within a defined area, utilizing data from radar and radio frequency direction finding (RF DF) sensors. Hence, the competition aims to identify new techniques focused on:

1. tracking the UAVs to show their location
2. indicating the type of object that has been detected (e.g. drone, bird)
3. indicating the type of UAV that has been observed (e.g. DJI Mavic Pro)

The evaluation of this competition considers the three aforementioned items while giving particular weight to the drone tracking task, which is scored according to how closely the results of the prediction algorithm match the true position of the UAV.

The work of this project focuses exclusively on the first task, i.e., on developing algorithms for predicting the UAV position, and leaves the other two tasks as future work.

3.1.1. Sensors

As previously mentioned, the NCIA dataset includes data from various types of sensors.

On the one hand, we can find data from two different **RF DF sensors**, which are able to provide classification and identification information about the UAV based on the RF signature. These sensors are:

- **Diana:** it is located at a latitude of 51.519137 deg and a longitude of 5.857951 deg. The antenna array of Diana is a linear combination of elements that is used to estimate range and bearing information about the target. However, it only reports elements detected in a 180 deg sector. The information provided by Diana has a high degree of uncertainty due to variations in the array antenna pattern.
- **Venus:** it is located at a latitude of 51.5192716 deg and a longitude of 5.8579155 deg. Unlike Diana, Venus only reports bearing information about the target, which has no ambiguity thanks to its usage of a circular array antenna.

On the other hand, we can find data from two different **radar sensors**, which provide information about the position (latitude, longitude, altitude), bearing and range of the detected UAVs. These sensors are:

- **Alvira:** it is located at a latitude of 51.52126391 deg and a longitude of 5.85862734 deg. Alvira is a 2D radar, but it is able to report an altitude value for the target.
- **Arcus:** it is located at a latitude of 51.52147 deg and a longitude of 5.87056833 deg. Arcus is a 3D sensor, so it should provide a more accurate report of the altitude of the UAV.

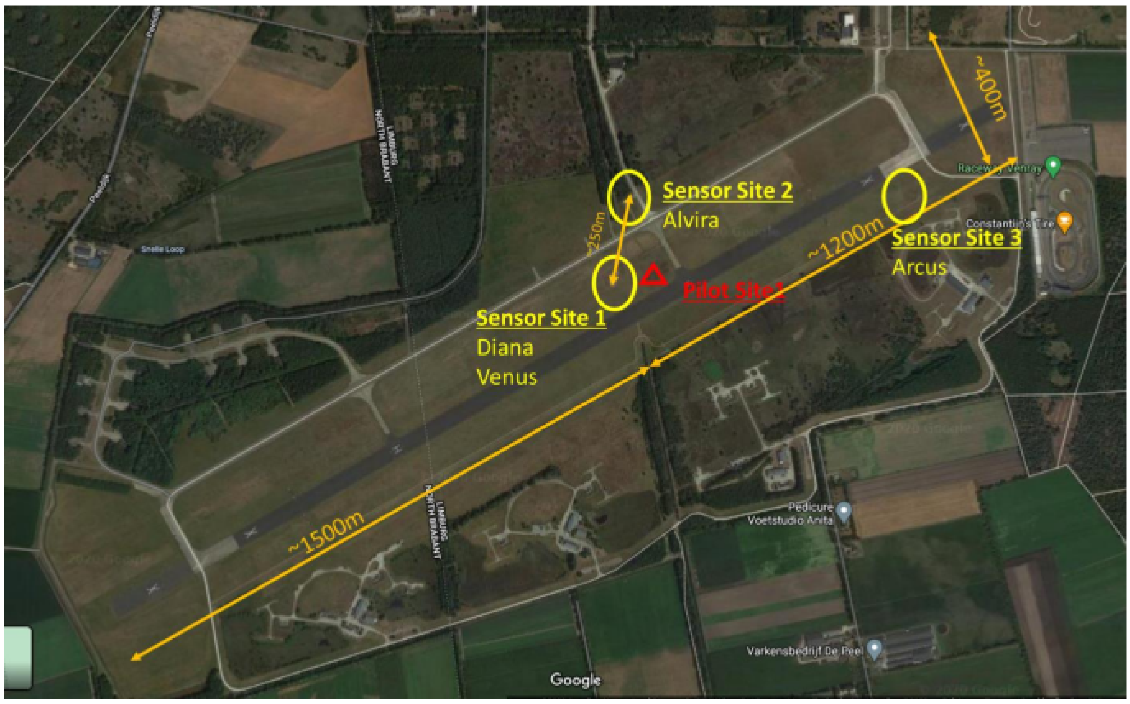


Figure 3.1: Sensor locations (figure taken from [1])

According to the description of the challenge [1], the measurements of the UAV were carried out at a C-UAS test centre in the Netherlands. A map of this location is shown in Figure 3.1, where the positions of the different sensors are marked. In this regard, we can see that the two RF DF sensors, Diana and Venus, are located close to each other at the centre of the flight zone (Sensor Site 1). The radar sensors are located at the edges of the flight zone: Alvira is located at Sensor Site 2, and Arcus is located at Sensor Site 3. All sensors were deployed at an elevation of about 31m.

The orange lines depict the approximate length between two points. In that sense, the line labeled as ∼1500m indicates the length between the beginning of the flight zone and Sensor Site 1. The length between Sensor Site 1 and the end of the flight zone is approximately 1200m. Moreover, the distance between Sensor Site 1 and Sensor Site 2 is 250 meters, and the width of the flight zone is 400 meters. Additionally, the red mark labeled as Pilot Site 1 shows the location from where the pilot controls the flight of the UAV.

3.1.2. Scenarios

The dataset of the ICMCIS drone challenge includes two types of scenarios, the ones using only one drone and others using two drones. Our work puts the focus exclusively on the scenarios that feature a single drone with various flight patterns. This is referred to as Scenario 1 and the corresponding sub-scenarios 1.1, 1.2, 1.3 and 1.4.

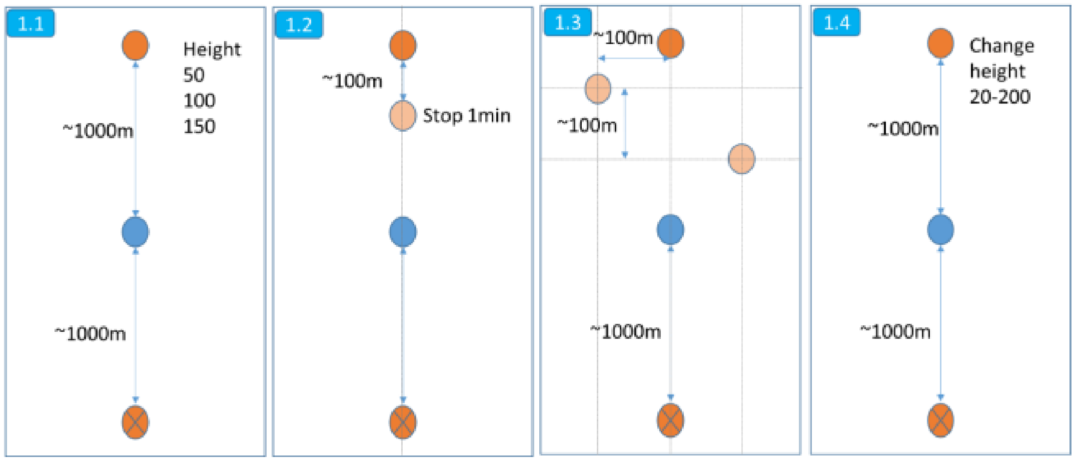


Figure 3.2: Scenario 1 flight missions (figure taken from [1])

Figure 3.2 shows the four sub-scenarios included in Scenario 1 of the NCIA dataset and which are part of NATO's description for this challenge. In these figures, the colored circles have the following meaning (from top to bottom):

- ● starting position of the mission
- ● stop points in the flight path of the UAV
- ● location of the RF DF sensors (Sensor Site 1)
- ⊗ end of the mission

As we can observe in Figure 3.2, there are some significant differences in the flight paths followed by the UAVs in each of the described sub-scenarios:

- In **Scenario 1.1**, the UAV moves from its starting position towards the final position, which is at an approximate distance of 2,000 meters. During its trajectory, the drone varies the height at which it flies.

- In **Scenario 1.2**, the UAV stops for one minute after it has traversed approximately 100 meters and then resumes its path towards the final position.
- In **Scenario 1.3**, the UAV first moves until stopping at a point 100 meters to the left of the starting position, then it continues moving forward until stopping at a point 100 meters to the right of the starting position, and finally it continues its path towards the final position.
- In **Scenario 1.4**, the UAV moves from its starting position to the final position while changing its altitude from 20 meters to 200 meters.

3.1.3. Drones

Furthermore, it should be noted that the data contained in the scenarios defined in the NCIA dataset was obtained by four different drones:

- DJI MAVIC PRO
- DJI MAVIC 2
- DJI Phantom 4 Pro
- Parrot DISCO

The first three drones are all *multi-copter* drones, meaning that they have multiple propellers. Conversely, the Parrot DISCO drone has a *fixed-wing* airframe, and thus it has different flight characteristics from the other drones.

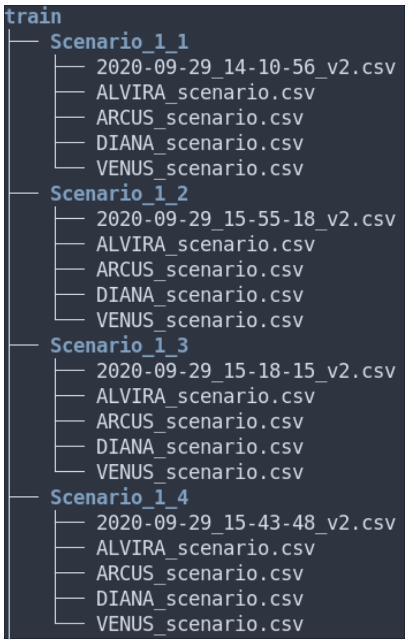
In addition, it should be mentioned that all the sensor data included in the NCIA dataset reflects readings of UAVs with a mass lower than 150 kg as, for instance, the typical hobby drones.

3.2 Data used in this project

The data of this challenge is organized in two folders, named `test` and `train`. In this project, we will only consider the data contained in the `train` folder since it contains ground truth files that can be used to evaluate the performance of the implemented models. In that sense, the data of this folder consists of multiple CSV files organized according to the scenario to which they belong. The file hierarchy of this folder can be observed in Figure 3.3.

The CSV files contain the following information:

- **Log files:** they are the files whose name is a date and time followed by the version (v2). These files provide the exact tracking information about the UAV, i.e., the 'true' position and velocity (ground truth) of the UAV at any given time.
- **Radar sensor data:** this data is contained in the files named `ALVIRA_scenario.csv` and `ARCUS_scenario.csv`. These files give the outputs from the radar sensors which detected the UAV during its flight. They provide location information (longitude, latitude, and altitude) as well as the bearing and range detected.



```
train
├── Scenario_1_1
│   ├── 2020-09-29_14-10-56_v2.csv
│   ├── ALVIRA_scenario.csv
│   ├── ARCUS_scenario.csv
│   ├── DIANA_scenario.csv
│   └── VENUS_scenario.csv
├── Scenario_1_2
│   ├── 2020-09-29_15-55-18_v2.csv
│   ├── ALVIRA_scenario.csv
│   ├── ARCUS_scenario.csv
│   ├── DIANA_scenario.csv
│   └── VENUS_scenario.csv
├── Scenario_1_3
│   ├── 2020-09-29_15-18-15_v2.csv
│   ├── ALVIRA_scenario.csv
│   ├── ARCUS_scenario.csv
│   ├── DIANA_scenario.csv
│   └── VENUS_scenario.csv
├── Scenario_1_4
│   ├── 2020-09-29_15-43-48_v2.csv
│   ├── ALVIRA_scenario.csv
│   ├── ARCUS_scenario.csv
│   ├── DIANA_scenario.csv
│   └── VENUS_scenario.csv
```

Figure 3.3: File hierarchy of the dataset

- **Radio frequency direction finding sensor data:** this data is contained in the files named `DIANA_scenario.csv` and `VENUS_scenario.csv`. These files show the outputs from the RF DF sensors which detected the UAV during its flight. They provide classification information based on kinematic and radar reflectivity characteristics, as well as identification data about the UAV based on the RF signature.

Given that this project aims to address the task of UAV tracking, we will only consider the data provided by the log files and the radar sensors since the RF DF data is aimed at the tasks of classification and identification of the UAV, which lie outside the scope to this project. Therefore, we will not consider the information reported by the sensors Diana and Venus.

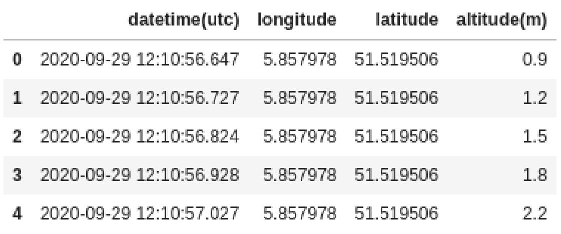
Moreover, since we are not going to address the tasks of identification and classification, we do not need to distinguish between the different types of drones described previously. Instead we will use all the data corresponding to the different *multi-copter* drones and focus on creating a model that is capable of tracking a generic *multi-copter* UAV.

Next, we will describe the data of the log files and radar sensor in more detail, using the files of the `Scenario_1_1` folder as an example.

3.2.1. Ground truth data

The `2020-09-29_14-10-56_v2.csv` file contains the **ground truth** of the flight of the UAV. This file has multiple columns reporting various characteristics of the UAV during its flight, such as the position (latitude, longitude, altitude), velocity, rotation (yaw, pitch, roll), and datetime. For the design of our prediction models, we will use the datetime and position columns, which are the variables that we can correlate to the data provided by the sensors.

An example of the values contained in those columns is shown in Figure 3.4, where we can see that the values of longitude and latitude remain the same while the altitude increases, an indication that the drone is ascending vertically.



	datetime(utc)	longitude	latitude	altitude(m)
0	2020-09-29 12:10:56.647	5.857978	51.519506	0.9
1	2020-09-29 12:10:56.727	5.857978	51.519506	1.2
2	2020-09-29 12:10:56.824	5.857978	51.519506	1.5
3	2020-09-29 12:10:56.928	5.857978	51.519506	1.8
4	2020-09-29 12:10:57.027	5.857978	51.519506	2.2

Figure 3.4: Example of values in the relevant columns of the ground truth file of Scenario 1.1

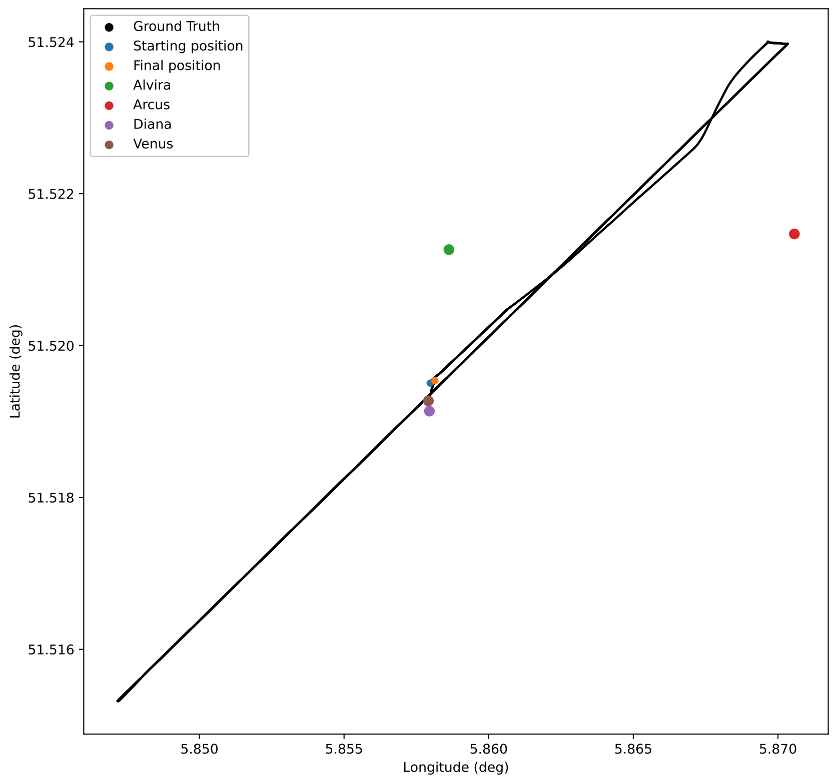


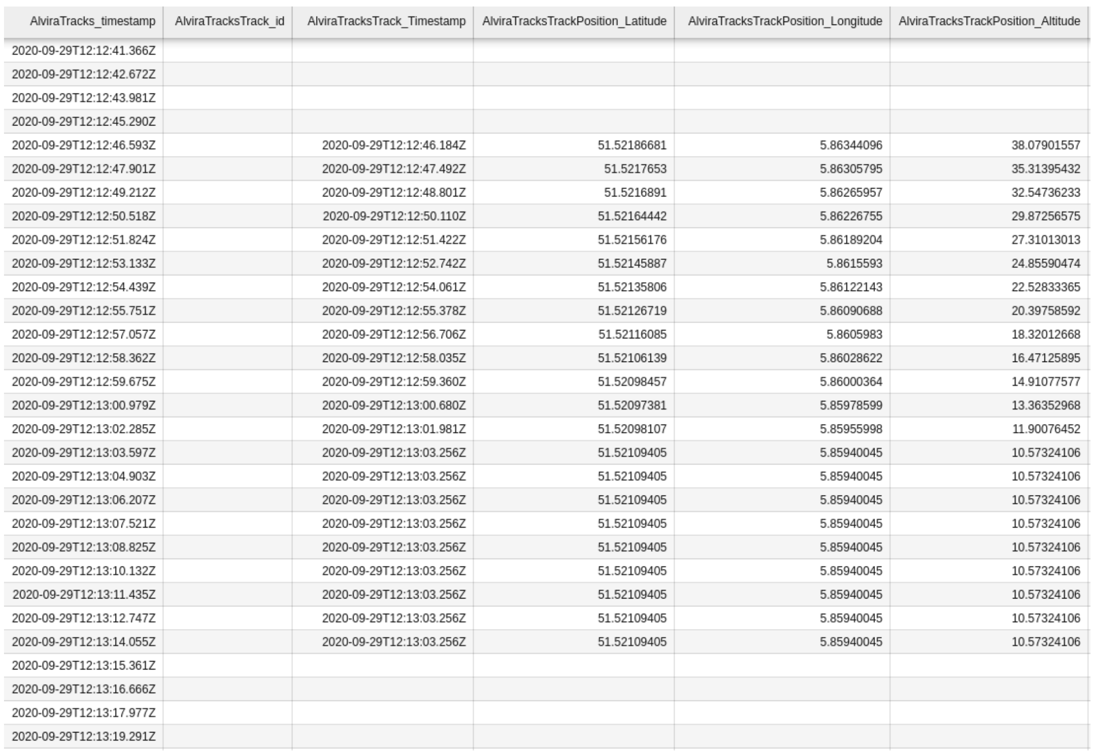
Figure 3.5: Position of the ground truth values and of the sensors in Scenario 1.1

As an illustration of the ground truth values of Scenario 1.1, Figure 3.5 shows the longitude and latitude of each of the reported values as well as the location of the radar sensors (Alvira and Arcus) and RF DF sensors (Diana and Venus). In addition, we have also marked the starting and final position of the UAV.

As can be seen in Figure 3.5, the UAV starts its flight path right next to the position of the RF DF sensors (Diana and Venus), and it starts moving towards the top right corner of the figure (starting position of the mission in Scenario 1.1 of Figure 3.2). Once it reaches the end of the flight zone, the UAV turns around and moves towards the bottom left corner of the figure (end of the mission in Scenario 1.1 of Figure 3.2). Finally, after reaching the end of the mission, it turns around once more and flies back towards its starting position, stopping very close to it (the flight back towards the starting position completely overlaps the outbound flight in Figure 3.5).

3.2.2. Radar sensor data

The `ALVIRA_scenario.csv` file contains the data reported by the Alvira sensor. This file has multiple columns with information relayed by the sensor for every timestamp and



AlviraTracks_timestamp	AlviraTracksTrack_id	AlviraTracksTrack_Timestamp	AlviraTracksTrackPosition_Latitude	AlviraTracksTrackPosition_Longitude	AlviraTracksTrackPosition_Altitude
2020-09-29T12:12:41.366Z					
2020-09-29T12:12:42.672Z					
2020-09-29T12:12:43.981Z					
2020-09-29T12:12:45.290Z					
2020-09-29T12:12:46.593Z		2020-09-29T12:12:46.184Z	51.52186681	5.86344096	38.07901557
2020-09-29T12:12:47.901Z		2020-09-29T12:12:47.492Z	51.5217653	5.86305795	35.31395432
2020-09-29T12:12:49.212Z		2020-09-29T12:12:48.801Z	51.5216891	5.86265957	32.54736233
2020-09-29T12:12:50.518Z		2020-09-29T12:12:50.110Z	51.52164442	5.86226755	29.87256575
2020-09-29T12:12:51.824Z		2020-09-29T12:12:51.422Z	51.52156176	5.86189204	27.31013013
2020-09-29T12:12:53.133Z		2020-09-29T12:12:52.742Z	51.52145887	5.8615593	24.85590474
2020-09-29T12:12:54.439Z		2020-09-29T12:12:54.061Z	51.52135806	5.86122143	22.52833365
2020-09-29T12:12:55.751Z		2020-09-29T12:12:55.378Z	51.52126719	5.86090688	20.39758592
2020-09-29T12:12:57.057Z		2020-09-29T12:12:56.706Z	51.52116085	5.8605983	18.32012668
2020-09-29T12:12:58.362Z		2020-09-29T12:12:58.035Z	51.52106139	5.86028622	16.47125895
2020-09-29T12:12:59.675Z		2020-09-29T12:12:59.360Z	51.52098457	5.86000364	14.91077577
2020-09-29T12:13:00.979Z		2020-09-29T12:13:00.680Z	51.52097381	5.85978599	13.36352968
2020-09-29T12:13:02.285Z		2020-09-29T12:13:01.981Z	51.52098107	5.85955998	11.90076452
2020-09-29T12:13:03.597Z		2020-09-29T12:13:03.256Z	51.52109405	5.85940045	10.57324106
2020-09-29T12:13:04.903Z		2020-09-29T12:13:03.256Z	51.52109405	5.85940045	10.57324106
2020-09-29T12:13:06.207Z		2020-09-29T12:13:03.256Z	51.52109405	5.85940045	10.57324106
2020-09-29T12:13:07.521Z		2020-09-29T12:13:03.256Z	51.52109405	5.85940045	10.57324106
2020-09-29T12:13:08.825Z		2020-09-29T12:13:03.256Z	51.52109405	5.85940045	10.57324106
2020-09-29T12:13:10.132Z		2020-09-29T12:13:03.256Z	51.52109405	5.85940045	10.57324106
2020-09-29T12:13:11.435Z		2020-09-29T12:13:03.256Z	51.52109405	5.85940045	10.57324106
2020-09-29T12:13:12.747Z		2020-09-29T12:13:03.256Z	51.52109405	5.85940045	10.57324106
2020-09-29T12:13:14.055Z		2020-09-29T12:13:03.256Z	51.52109405	5.85940045	10.57324106
2020-09-29T12:13:15.361Z					
2020-09-29T12:13:16.666Z					
2020-09-29T12:13:17.977Z					
2020-09-29T12:13:19.291Z					

Figure 3.6: Example of values in the relevant columns of the Alvira file of Scenario 1.1

some specific columns that have values only when this sensor detects a target, such as its position (latitude, longitude, altitude), speed, azimuth, and elevation.

According to the challenge description [1], the columns whose name contains the string "TracksTrack" are the ones that should be considered as the measurements reported by the Alvira sensor. Taking this into account, we selected the columns describing the timestamp and the location reported by the sensor.

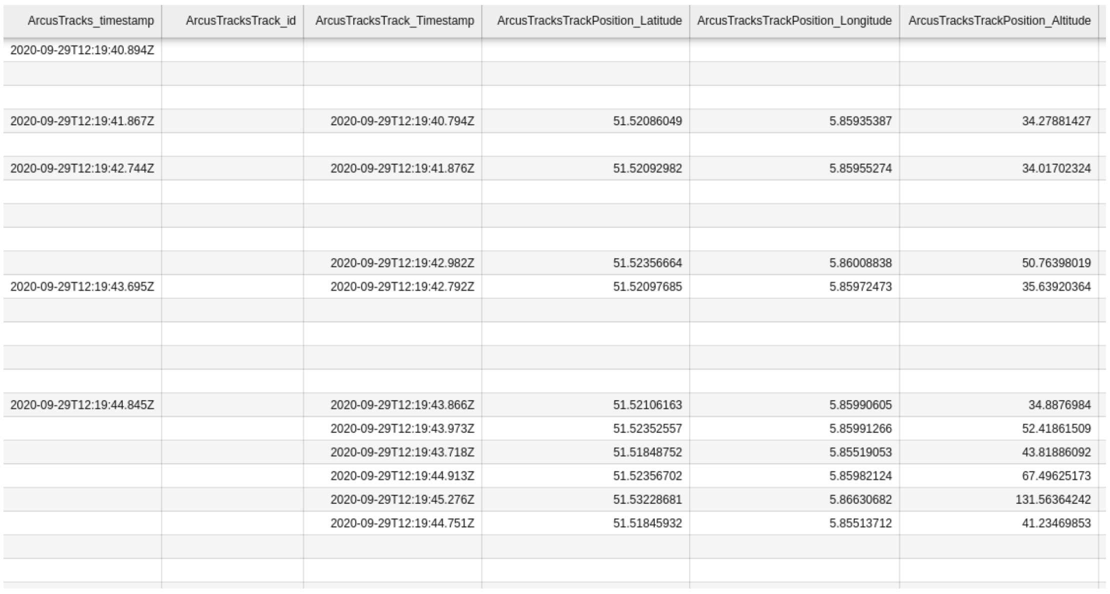
An example of the values contained in those columns is shown below in Figure 3.6. As it can be seen in such figure, the sensor only reports information when a target is detected, and so there are gaps in the reported data.

The `ARCUS_scenario.csv` file contains the data reported by the Arcus sensor. This file has similar columns to the file of Alvira, with the general columns relayed by the sensor at every timestamp and the specific ones that have values only when the sensor detects a target, such as its position (latitude, longitude, altitude), speed, azimuth, and elevation.

Just like with the Alvira sensor, here, the columns whose name contains the string "TracksTrack" are the ones that should be considered as the measurements reported by the sensor. Taking this into account, we again selected the columns describing the timestamp and the location reported by the sensor.

An example of the values contained in those columns is shown below in Figure 3.7. As it can be seen in that figure, similarly to Alvira, this sensor only reports data when a target is detected, and so there are gaps in the reported data.

However, when comparing Figures 3.6 and 3.7 we can observe that the readings of Arcus differ from those of Alvira in that they contain intermittent gaps. This is most likely due to the fact that Arcus is a 3D radar sensor, and therefore it has the added challenge of detecting the depth of targets. The complexity of that challenge, combined with the fact that the resolution of the sensor diminishes with the distance to the detected target, would thus be the reason for it to report sparse data.



ArcusTracks_timestamp	ArcusTracksTrack_id	ArcusTracksTrack_Timestamp	ArcusTracksTrackPosition_Latitude	ArcusTracksTrackPosition_Longitude	ArcusTracksTrackPosition_Altitude
2020-09-29T12:19:40.894Z					
2020-09-29T12:19:41.867Z		2020-09-29T12:19:40.794Z	51.52086049	5.85935387	34.27881427
2020-09-29T12:19:42.744Z		2020-09-29T12:19:41.876Z	51.52092982	5.85955274	34.01702324
		2020-09-29T12:19:42.982Z	51.52356664	5.86008838	50.76398019
2020-09-29T12:19:43.695Z		2020-09-29T12:19:42.792Z	51.52097685	5.85972473	35.63920364
2020-09-29T12:19:44.845Z		2020-09-29T12:19:43.866Z	51.52106163	5.85990605	34.8876984
		2020-09-29T12:19:43.973Z	51.52352557	5.85991266	52.41861509
		2020-09-29T12:19:43.718Z	51.51848752	5.85519053	43.81886092
		2020-09-29T12:19:44.913Z	51.52356702	5.85982124	67.49625173
		2020-09-29T12:19:45.276Z	51.53228681	5.86630682	131.56364242
		2020-09-29T12:19:44.751Z	51.51845932	5.85513712	41.23469853

Figure 3.7: Example of values in the selected columns of the Arcus file of Scenario 1.1

3.2.3. Data preprocessing

In order to be able to harness the information of the NCIA dataset through the different models that will be explained in the following sections, it was necessary to execute certain preprocessing steps.

Firstly, we selected the columns directly concerning the task of trajectory tracking (timestamps and location), as explained in the previous section. Moreover, since this information is not available for every time step of the sensor records, we discarded the empty rows for those columns.

Afterwards, we increased the values in the altitude column of the files containing data from the sensors by 31 meters, which is the altitude at which the sensors are located. As a result, we obtained the true altitude of the detected UAV and were able to compare it to the one reported in the ground truth files.

Furthermore, since the datetime format of the sensor files and the ground truth files are different, we converted the timestamp information of the sensors from the TZ format to the UTC format of the ground truth in order to be able to draw comparisons of the time instants between those files.

All in all, Table 3.1 shows the number of data samples obtained from each source after the described preprocessing steps. As it can be seen in the table, the Arcus sensor is the one that provides the majority of the data samples, while Alvira contributes with a much smaller percentage of samples. It should also be noted that the ground truth files provide significantly fewer data values than the sensor data files.

Given that we want to consider all the data reported by the sensors to obtain the predictions, in this project, we combined and sorted the data reported by both Alvira and Arcus according to the timestamp values. We will refer to this combined data as sensor data throughout this project.

3.2.4. Algorithm for correspondence

At several points during the development of this project, we needed to establish a correspondence between the ground truth samples with either the sensor data samples or

Scenario	Data source	Number of samples
Scenario 1.1	Ground Truth	7,979
	Alvira	162
	Arcus	16,965
Scenario 1.2	Ground Truth	8,371
	Alvira	253
	Arcus	134,805
Scenario 1.3	Ground Truth	4,729
	Alvira	138
	Arcus	43,891
Scenario 1.4	Ground Truth	5,652
	Alvira	270
	Arcus	85,805

Table 3.1: Number of samples in each file of the dataset after preprocessing

the obtained predictions. This was the case, for instance, when computing the errors of the prediction process. However, the significant difference in the number of samples in the ground truth set and in the combined sensor data (and consequently in the obtained predictions), as described in Table 3.1, made it difficult to establish such correspondence.

Therefore, we developed an algorithm that chronologically analyzes each ground truth sample, and associates the *i-th* sample g_i with the sensor data sample whose timestamp is closer to the timestamp of g_i (also considering the sensor data in chronological order). The algorithm works by calculating the absolute difference between the timestamp of a given g_i and the timestamp of the *j-th* sensor data sample s_j, which we will refer to as $d_{i,j}$. Then, it computes the absolute difference between the timestamp of g_i and the timestamp of the next sensor data sample s_{j+1}, which we represent by $d_{i,j+1}$. If $d_{i,j+1}$ is smaller than $d_{i,j}$, then it means that s_{j+1} is closer in time to the ground truth sample g_i than s_j, and so s_j can be discarded. We repeat this process for the next sensor data sample s_{j+2}, and so on. Conversely, if $d_{i,j+1}$ is larger than $d_{i,j}$, we know that s_j is the sensor sample closest in time to the ground truth sample. In such a case, we associate g_i with s_j and repeat the same process for the next ground truth sample g_{i+1} from the sensor data sample s_{j+1}.

It should also be mentioned that if when considering a certain g_i and s_j it occurs that s_j is the last sensor data sample (in chronological order), and so we cannot compare g_i with any other samples, the algorithm by default associates the last sensor data sample s_j to the ground truth sample g_i. Therefore, the algorithm considers two different stopping conditions, which are either when every ground truth sample matches a sensor data sample or when we run out of sensor data samples.

3.3 Metrics for evaluating the tracking problem

As previously mentioned in Section 2.1, this project deals with the problem of tracking a moving UAV during its flight. Different sources of data about the UAVs are mentioned in such section, but according to the characteristics of the data contained in the NCIA dataset, we will specifically be relying on the data reported by radar sensors.

In that sense, we will develop models that are able to predict the state of the UAV at a given time, considering the previous states of the UAV as reported by the radar sensors. The state of the UAV will be defined by the three variables that describe its position according to the data in consideration, i.e. longitude, latitude, and altitude.

In order to evaluate the performance of the different models that will be presented later in the project, we will use various metrics to measure the errors of the prediction process. Next, we will describe these metrics. We will denote the predicted value at timestamp i by $\hat{y}_i$, the true value by y_i and we will use N to represent the total number of pairs of predicted and true values considered.

In this project, we will utilize the Mean Absolute Error (MAE), the Mean Squared Error (MSE), and the Normalized Root Mean Squared Error (NRMSE), defined respectively as:

$$\text{MAE} = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i| \tag{3.1}$$

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2 \tag{3.2}$$

$$\text{NRMSE} = \frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2}}{\bar{y}} \tag{3.3}$$

It should be noted that the previously described metrics will be computed independently for each of the state variables of the problem, that is, for the longitude, latitude, and altitude.

On the other hand, the challenge associated with the NCIA dataset [1] proposed a custom metric in order to evaluate the submitted proposals. We will also consider this metric, which is defined as:

$$\begin{aligned}&\text{NCIAMetric} = \\ &= \frac{1}{N}\sum_{j=1}^{N}\left(pp\cdot\sqrt{\alpha(\hat{lat}-lat)^2+\alpha(\hat{lon}-lon)^2+(\hat{alt}-alt)^2}+pc\cdot|\hat{c}-c|+pi\cdot|\hat{i}-i|\right)\end{aligned}$$

Where:

- N is the number of rows in the solution submitted
- $a = 100000$ is the scaling factor of longitude and latitude degree
- $\hat{lat}, \hat{lon}, \hat{alt}$ are the predicted locations for a given timestamp
- lat, lon, alt are the ground truth locations for a given timestamp
- $pp = 0.7$ is the position penalty
- $\hat{c}$, c are the predicted and ground truth classes. This modulo difference is 0 if the class is correctly predicted and 1 otherwise. Since we aren't performing the task of classification, we will set this difference to 1 (wrong class).
- $pc = 0.15$ is the classification penalty

- $\hat{i}$, i are the predicted and ground truth identities. This modulo difference is 0 if the identity is correctly predicted and 1 otherwise. Since we aren't performing the task of identification, we will set this difference to 1 (wrong identity).
- $pi = 0.15$ is the identification penalty

In order to determine how good our results are according to this metric, we will consider as a reference the scores of the top four teams in the challenge [1] as evaluated by the NCIAMetric:

- Team 1: Paschalina Medentzidou_Submission_v2 = 9.325
- Team 2: CERTH ITI Varlab Team_Submission_v4 = 10.589
- Team 3: Horizon Team_Submission_v1= 10.447
- Team 4: DSTL Team_Submission_v1 = 22.101

It should be noted, however, that these are the accuracy results obtained by the participant teams for the *unknown testing sets* that were run by the organization of the challenge, and thus they are not directly comparable to those obtained throughout this project.

CHAPTER 4

Prediction model with Kalman Filters

The Kalman Filter is an algorithm for estimating the unknown variables of a dynamic system based on a series of measurements reported by various sensors. The reason that motivated us to use Kalman Filters for predicting the trajectory of a UAV is that they allow modeling the parameters that define the learning of the underlying dynamics of the system.

This chapter is structured as follows. Section 4.1 provides a theoretical overview of Kalman Filters. Section 4.2 defines the way in which we formulated the problem and how we designed the model in order to obtain experimental results. Finally, Section 4.3 shows the results (and the corresponding analysis) obtained by applying the linear Kalman Filter model to the data obtained from the NCIA dataset [1].

4.1 An overview of linear Kalman Filters

This section is intended to provide an overview of the key design principles of Kalman Filters. While the literature on this topic is very abundant, our intention is to discuss only the main notions needed to design a Kalman Filter. For this purpose, our exposition is based on an excellent tutorial on Kalman Filters written by the researcher Alex Becker [2].

First of all, we should mention that in this project, we will only consider linear Kalman Filters. There exist, however, some more complex versions of this algorithm that introduce non-linearities in the process model and in the observation model, such as the Extended Kalman Filter or the Unscented Kalman Filter.

Our challenge is to track the trajectory of a UAV and provide accurate and precise estimations of the UAV position in terms of three hidden variables (longitude, latitude and altitude) using the received measurements of the observed variables as reported by the radar sensors (Alvira and Arcus). The sensor readings are characterized by the presence of uncertainty due to miscalibration, the precision of the receiver clock, range degradation, etc. Consequently, the use of Kalman Filters is very appropriate as it is one of the best algorithms to produce estimates of the hidden variables based on inaccurate and uncertain measurements.

The behaviour of radar sensors when tracking a target consists in sending a beam in the direction of the target (UAV) every t seconds and estimating the current target's position and velocity (and possibly acceleration). Moreover, utilizing that information, the sensors predict the position of the target at the next track beam.

In that sense, the future position of the UAV can be predicted using the following kinematic equation:

$$x = x_0 + \dot{x}_0 \Delta t + \frac{1}{2} \ddot{x} \Delta t^2$$

where:

- x is the UAV's position
- x_0 is the initial position of the UAV
- $\dot{x}_0$ is the initial velocity of the UAV
- $\ddot{x}$ is the UAV's acceleration
- Δt is the time interval

Since we are working with three variables, we will have three kinematic equations, one for each dimension (longitude, latitude, altitude). The vector $[x, y, z, \dot{x}, \dot{y}, \dot{z}, \ddot{x}, \ddot{y}, \ddot{z}]$ is called the **system state** and represents the target parameters (x is the longitude dimension, y is the latitude dimension, and z is the altitude dimension).

The three kinematic equations define the **dynamic model**, which describes the relationship between the current state (input) and the next target state (output). However, as we said before, the uncertainty in the calibration, beam width, or return echo of the sensors produces a **measurement noise** that introduces imperfections in the measurements reported by the sensors in relation to the true state of the UAV.

Additionally, the motion of the UAV may not be perfectly aligned to the kinematic equations of the dynamic model due to external factors such as atmospheric effects (wind, air turbulence, etc.) or pilot maneuvers. The uncertainty that these factors introduce in the dynamic model is called the **process noise**.

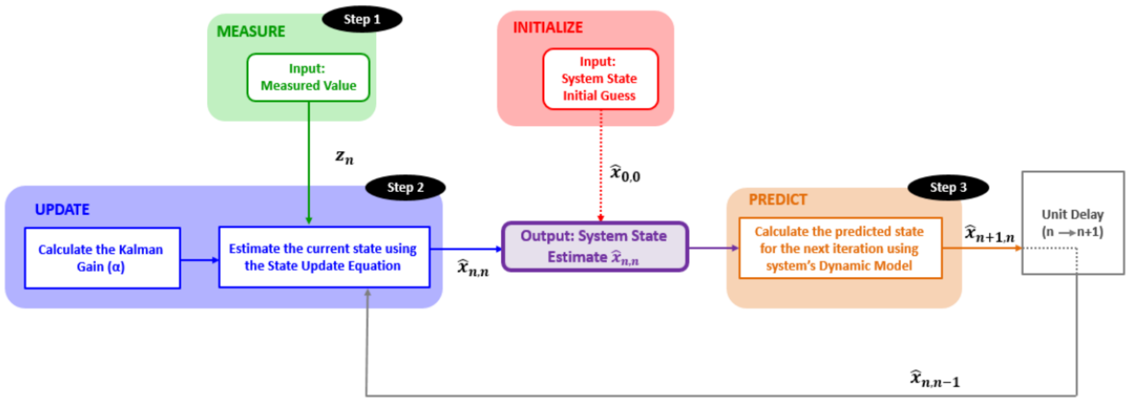

MEASURE
Step 1
Input: Measured Value
z_n
INITIALIZE
Input: System State Initial Guess
$\hat{x}_{0,0}$
UPDATE
Step 2
Calculate the Kalman Gain (α)
Estimate the current state using the State Update Equation
$\hat{x}_{n,n}$
Output: System State Estimate $\hat{x}_{n,n}$
PREDICT
Step 3
Calculate the predicted state for the next iteration using system's Dynamic Model
$\hat{x}_{n+1,n}$
Unit Delay (n → n+1)
$\hat{x}_{n,n-1}$

Figure 4.1: Kalman Filter estimation algorithm (image taken from [2])

Figure 4.1 depicts the general behaviour of the Kalman Filter estimation algorithm. It consists of a series of iterations aimed at two main operations:

1. calculate the current system state estimate $\hat{x}_{n,n}$ ($\hat{x}_{n,n}$ denotes the estimate calculated at time n of the system state at time n).
2. predict the future system state $\hat{x}_{n+1,n}$ using the dynamic model of the system ($\hat{x}_{n+1,n}$ denotes the prediction computed at time n of the system state at time $n+1$).

In the initialization iteration, the algorithm uses the initial guess of the system state $\hat{x}_{0,0}$ and predicts the next state $\hat{x}_{1,0}$. In the subsequent iterations, the input to the algorithm is the measurements from the sensors (z_n values in Step 1) and the predicted state from the previous iteration that now becomes the previous estimate in the current iteration. From these values, the algorithm estimates the current state using the **state update equation** (Step 2) and predicts the next state using the **prediction equation** or **state extrapolation equation** (Step 3). It should be mentioned as well that in Step 2, the Kalman Gain, K_n, which is a factor in the state update equation, is dynamically computed for each iteration of the filter.

As an example, in the first iteration the system receives the first measurement from the sensors, z_1, and the previous estimate, $\hat{x}_{1,0}$, and calculates the current estimate, $\hat{x}_{1,1}$, and the prediction of the future state $\hat{x}_{2,1}$.

The key idea of the Kalman Filter algorithm is that it successively updates the system state by adjusting the difference between the prediction done at time $n-1$ of the system state at time n and the measurement of the system at time n.

4.1.1. Example: tracking an aircraft

In this section, we show an illustrative example in order to facilitate the understanding of the Kalman Filter algorithm. For the sake of simplicity, and in order to keep the explanation as straightforward as possible, we will adopt the following assumptions:

- two target parameters (position and velocity)
- constant velocity
- one dimension; i.e., we will assume only one variable to represent the true value of the aircraft position (note that in our problem we use three variables: longitude, latitude, altitude)
- the Kalman Gain factors in the state update equations (α, β) will be constants instead of being dynamically computed in each iteration

In the following section 4.2, we will relax these assumptions for more accurate modeling of the UAV in the context of the problem that concerns us.

Since we are working in this example with two target parameters, we are going to track the trajectory of the aircraft using an $\alpha-\beta$ filter. This is a type of filter very closely related to the Kalman Filter, but the difference lies in considering manually selected and static values for the factors of the state update equations. In that sense, in our case, the filter will have an α parameter for the equation of the position and a β parameter for the equation of the velocity.

Figure 4.2 graphically shows the problem of tracking an aircraft that we are trying to solve with this approximated model. The main variables of the system are:

1. x_n: the position to the aircraft at time n.
2. $\dot{x} = v = dx/dt$: the velocity is a derivative of the aircraft position.
3. Δt: the tracking time interval.

The dynamic model or **state extrapolation equations** of motion are:

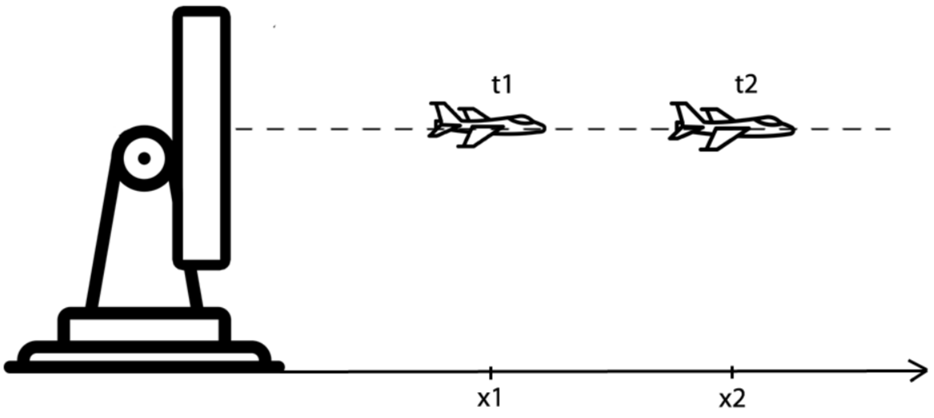


Figure 4.2: Tracking an aircraft (image taken from [2])

- $x_{n+1} = x_n + \Delta t \dot{x}_n$: the position at the next iteration is equal to the position at the current cycle plus the velocity of the target multiplied by the time interval
- $\dot{x}_{n+1} = \dot{x}_n$: the velocity is constant

And the $\alpha - \beta$ **update state equations** are:

- The update state equation for the position is $\hat{x}_{n,n} = \hat{x}_{n,n-1} + \alpha(z_n - \hat{x}_{n,n-1})$: the updating considers the difference between the prediction of x_n at time $n-1$ ($\hat{x}_{n,n-1}$) and the sensor measurement at time n (z_n).
- The update state equation for the velocity is $\hat{\dot{x}}_{n,n} = \hat{\dot{x}}_{n,n-1} + \beta((z_n - \hat{x}_{n,n-1})/\Delta t)$

4.1.2. The uncertainty

In this subsection, we will intuitively analyze how the uncertainty is handled in the Kalman Filter algorithm.

As it can be seen in the update state equations, the parameters α and β are the weights given to the measurements reported by the sensors (z_n), while $(1 - \alpha)$ and $(1 - \beta)$ are the weights given to the estimates (i.e., $\hat{x}_{n,n-1}$). This can be observed for instance in the update state equation of the position:

$$\hat{x}_{n,n} = \hat{x}_{n,n-1} + \alpha(z_n - \hat{x}_{n,n-1}) = \hat{x}_{n,n-1} + \alpha z_n - \alpha \hat{x}_{n,n-1} = \alpha z_n + (1 - \alpha)\hat{x}_{n,n-1}$$

Therefore, since the values of the α and β parameters are manually selected, we should take into account the uncertainty of the measurement and the uncertainty associated with the estimation process in order to select values that optimize the performance of the algorithm.

In the case of the Kalman Filter, those factors are called Kalman Gain (K_n) (see the blue box in Figure 4.1), and their value is within the interval $[0, 1]$. Just like in the case of the α-β filter, K_n is the weight we give to the sensor measurement, while $(1 - K_n)$ is the weight given to the estimate. When the uncertainty of the measurement is very large and the estimate uncertainty is very small, K_n is close to 0. In the opposite case, when K_n is close to 1, we are giving a large weight to the measurement and a small weight to the estimate.

The main characteristic of the Kalman Gain is that it is dynamically calculated at each iteration of the filter taking into account the specified uncertainty associated with the measurement process and the extrapolated uncertainty of the estimation process.

Therefore, besides the input measurement values, the Kalman Filter algorithm requires a parameter that represents the measurement uncertainty. The value of this parameter is typically provided by the equipment vendor or by the results of the equipment calibration, which in the case of the radar measurement will depend on the beam width, clock stability, etc. The standard notation for the measurement uncertainty or **measurement noise** is the letter r (or matrix R).

Moreover, the Kalman Filter also requires an initial value for the estimate's uncertainty, which will then be updated in the different iterations of the algorithm. This estimation uncertainty is typically represented by the letter p (or matrix P, which is typically called Covariance Matrix).

We also need to take into account that there are uncertainties associated with the dynamic model of the system. For instance, we assumed a constant velocity in the example presented in section 4.1.1; however, it is likely this is not the case due to possible aircraft maneuvers. The uncertainty of the dynamic model of the system is referred to as **process noise**. The standard notation for the process noise is by the letter q (or matrix Q, also called Predictor Covariance Equation).

4.2 Problem formulation with a linear Kalman Filter

In this project, we used the Python library `FilterPy` [15] to define the linear Kalman Filter model. This library implements various types of Bayesian Filters, and it is geared towards a pedagogical purpose, not sacrificing clarity over speed.

Firstly, it should be noted that we are tracking in three dimensions, as that is the dimensionality of the reported data by the sensors, and so we will have three *observed variables* x, y, and z, which respectively correspond to the longitude, latitude, and altitude. This means that the measurements that the filter will receive can be represented as:

$$z = \begin{bmatrix} x & y & z \end{bmatrix}^T \tag{4.1}$$

In order to implement a Kalman Filter, the first design decision that we had to make was the selection of the **state variables**. After analyzing the trajectory and the data of the ground truth, it became obvious that neither the velocity nor the acceleration were constant, as there were changes in the velocity of the UAV and also in the rate of the change of its velocity. This can be observed, for instance, in Figure 4.3, where we can see a plot of the velocity in each dimension for every sample in the ground truth of Scenario 1.1. We are considering that velocityX is the velocity that corresponds to the longitude, velocityY corresponds to the latitude, and velocityZ corresponds to the altitude.

We can see in Figure 4.3 that in every dimension the value of the velocity changes in a non-uniform way, having periods when it does not change at all, or it does so very slightly, and other periods when we can see a peak or a valley corresponding to a sudden increase and decrease of the velocity in that dimension. It should be mentioned that these sudden changes mainly occur for the longitude (velocityX) and latitude (velocityY), since, as expected, there are not too many sudden acceleration or deceleration moments related to the altitude (velocityZ) of the UAV.

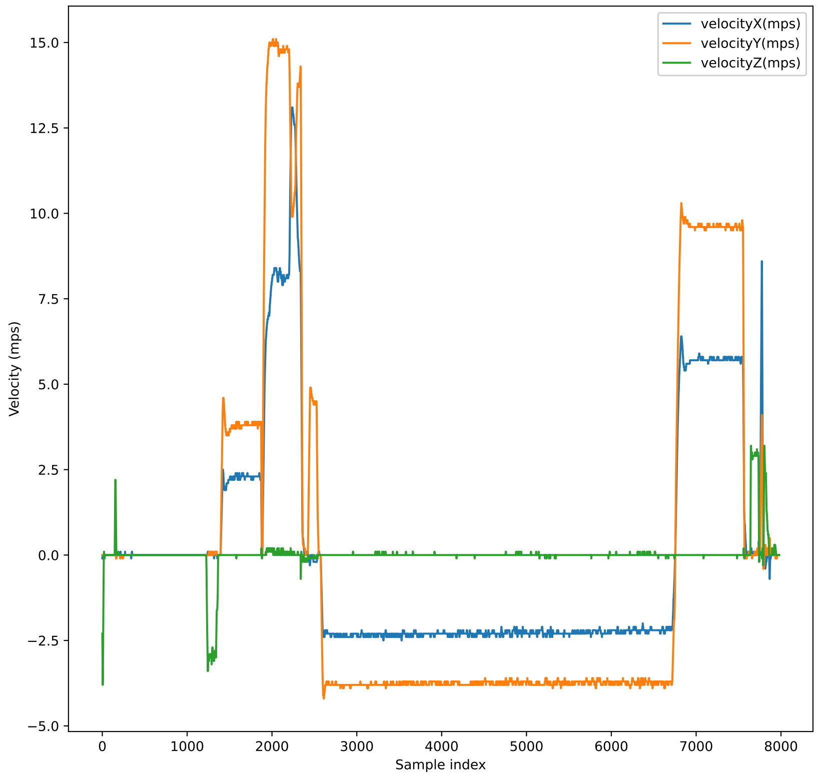


Figure 4.3: Velocity in each dimension for every sample in the ground truth of Scenario 1.1

All in all, these irregular patterns led us to believe that neither the velocity, nor its rate of change (acceleration) can be considered constant when modelling the behaviour of the UAV in this project.

There have also been mentions in the literature that refer to the non-constant acceleration that UAVs present in their flight as reported by on-board accelerometers. Such information can, for instance, be found in [16].

Taking into account these considerations, we assume a model of non-constant acceleration dynamics and define the state vector of the Kalman Filter as a 12-dimensional feature vector:

$$\boldsymbol{x} = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} & \ddot{x} & \ddot{y} & \ddot{z} & \dddot{x} & \dddot{y} & \dddot{z} \end{bmatrix}^T \tag{4.2}$$

where we can find the following components:

- x, y, z: the observed variables (longitude, latitude, altitude)
- $\dot{x}, \dot{y}, \dot{z}$: the velocity of the three observed variables
- $\ddot{x}, \ddot{y}, \ddot{z}$: the acceleration of the three observed variables
- $\dddot{x}, \dddot{y}, \dddot{z}$: the derivative of the acceleration of the three observed variables

Once the important design decision of selecting the state variables is made, we are ready to use the `FilterPy` [15] library to construct an object class `KalmanFilter` by specifying the dimensionality of the state vector and of the measurement data the process will have.

The `KalmanFilter` object will have default values assigned to the different matrices that define the behaviour of the filter. However, it is recommended to assign these matrices values according to the specific task at hand in order to obtain the best possible performance.

In that sense, taking into account the specified state vector, we first defined the **State Transition Function *F*** according to Newtonian kinematics of third order, as implemented by the `kinematic_kf` function contained in the `FilterPy` [15] library. Thus, the state equations that define the dynamic model are the following:

$$
\begin{aligned}
x^- &= 1\cdot x + 0\cdot y + 0\cdot z + \Delta t\cdot \dot{x} + 0\cdot \dot{y} + 0\cdot \dot{z} + \tfrac{1}{2!}\Delta t^2\cdot \ddot{x} + 0\cdot \ddot{y} + 0\cdot \ddot{z} + \tfrac{1}{3!}\Delta t^3\cdot \dddot{x} + 0\cdot \dddot{y} + 0\cdot \dddot{z}\\
y^- &= 0\cdot x + 1\cdot y + 0\cdot z + 0\cdot \dot{x} + \Delta t\cdot \dot{y} + 0\cdot \dot{z} + 0\cdot \ddot{x} + \tfrac{1}{2!}\Delta t^2\cdot \ddot{y} + 0\cdot \ddot{z} + 0\cdot \dddot{x} + \tfrac{1}{3!}\Delta t^3\cdot \dddot{y} + 0\cdot \dddot{z}\\
z^- &= 0\cdot x + 0\cdot y + 1\cdot z + 0\cdot \dot{x} + 0\cdot \dot{y} + \Delta t\cdot \dot{z} + 0\cdot \ddot{x} + 0\cdot \ddot{y} + \tfrac{1}{2!}\Delta t^2\cdot \ddot{z} + 0\cdot \dddot{x} + 0\cdot \dddot{y} + \tfrac{1}{3!}\Delta t^3\cdot \dddot{z}\\
\dot{x}^- &= 0\cdot x + 0\cdot y + 0\cdot z + 1\cdot \dot{x} + 0\cdot \dot{y} + 0\cdot \dot{z} + \Delta t\cdot \ddot{x} + 0\cdot \ddot{y} + 0\cdot \ddot{z} + \tfrac{1}{2!}\Delta t^2\cdot \dddot{x} + 0\cdot \dddot{y} + 0\cdot \dddot{z}\\
\dot{y}^- &= 0\cdot x + 0\cdot y + 0\cdot z + 0\cdot \dot{x} + 1\cdot \dot{y} + 0\cdot \dot{z} + 0\cdot \ddot{x} + \Delta t\cdot \ddot{y} + 0\cdot \ddot{z} + 0\cdot \dddot{x} + \tfrac{1}{2!}\Delta t^2\cdot \dddot{y} + 0\cdot \dddot{z}\\
\dot{z}^- &= 0\cdot x + 0\cdot y + 0\cdot z + 0\cdot \dot{x} + 0\cdot \dot{y} + 1\cdot \dot{z} + 0\cdot \ddot{x} + 0\cdot \ddot{y} + \Delta t\cdot \ddot{z} + 0\cdot \dddot{x} + 0\cdot \dddot{y} + \tfrac{1}{2!}\Delta t^2\cdot \dddot{z}\\
\ddot{x}^- &= 0\cdot x + 0\cdot y + 0\cdot z + 0\cdot \dot{x} + 0\cdot \dot{y} + 0\cdot \dot{z} + 1\cdot \ddot{x} + 0\cdot \ddot{y} + 0\cdot \ddot{z} + \Delta t\cdot \dddot{x} + 0\cdot \dddot{y} + 0\cdot \dddot{z}\\
\ddot{y}^- &= 0\cdot x + 0\cdot y + 0\cdot z + 0\cdot \dot{x} + 0\cdot \dot{y} + 0\cdot \dot{z} + 0\cdot \ddot{x} + 1\cdot \ddot{y} + 0\cdot \ddot{z} + 0\cdot \dddot{x} + \Delta t\cdot \dddot{y} + 0\cdot \dddot{z}\\
\ddot{z}^- &= 0\cdot x + 0\cdot y + 0\cdot z + 0\cdot \dot{x} + 0\cdot \dot{y} + 0\cdot \dot{z} + 0\cdot \ddot{x} + 0\cdot \ddot{y} + 1\cdot \ddot{z} + 0\cdot \dddot{x} + 0\cdot \dddot{y} + \Delta t\cdot \dddot{z}\\
\dddot{x}^- &= 0\cdot x + 0\cdot y + 0\cdot z + 0\cdot \dot{x} + 0\cdot \dot{y} + 0\cdot \dot{z} + 0\cdot \ddot{x} + 0\cdot \ddot{y} + 0\cdot \ddot{z} + 1\cdot \dddot{x} + 0\cdot \dddot{y} + 0\cdot \dddot{z}\\
\dddot{y}^- &= 0\cdot x + 0\cdot y + 0\cdot z + 0\cdot \dot{x} + 0\cdot \dot{y} + 0\cdot \dot{z} + 0\cdot \ddot{x} + 0\cdot \ddot{y} + 0\cdot \ddot{z} + 0\cdot \dddot{x} + 1\cdot \dddot{y} + 0\cdot \dddot{z}\\
\dddot{z}^- &= 0\cdot x + 0\cdot y + 0\cdot z + 0\cdot \dot{x} + 0\cdot \dot{y} + 0\cdot \dot{z} + 0\cdot \ddot{x} + 0\cdot \ddot{y} + 0\cdot \ddot{z} + 0\cdot \dddot{x} + 0\cdot \dddot{y} + 1\cdot \dddot{z}
\end{aligned}
$$

An important decision related to these equations is the value of the timestep Δt. The problem we faced in that regard is that the value should be constant, but the data provided by the sensors do not have a constant timestep value. In the end, we decided to consider the combined data from both radar sensors and compute the difference between every two consecutive timestamps. Afterwards, we calculated the average of those values in order to obtain an approximate value for the timestep Δt.

Next, we used the `Q_continuous_white_noise` function of the library to define the **Process Noise Matrix *Q*** according to the Discretized Continuous White Noise Model. In order to refine this implementation, we decided to explore multiple values of the `spectral_density` parameter of that function, settling for a value of $5e^{-7}$, which we observed provided the best performance.

Afterwards, we defined the **Observation Matrix *H*** as follows:

$$
\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.3}
$$

The reason for this definition is derived from the usage of this matrix, which is defined by the equation $\boldsymbol{z} = \boldsymbol{Hx}$. Thus, the purpose of this matrix is to transform the state vector into a measurement vector. Given that in our specific case there is no need for any type of scaling factor to be applied when performing that conversion, we decided to use the default value of one in the diagonal.

Later, we modeled the noise of the sensors by assigning a value to the **Measurement Noise Matrix *R***, which is defined as follows:

$$\boldsymbol{R} = \begin{bmatrix} \sigma_x^2 & \sigma_y\sigma_x & \sigma_z\sigma_x \\ \sigma_x\sigma_y & \sigma_y^2 & \sigma_z\sigma_y \\ \sigma_x\sigma_z & \sigma_y\sigma_z & \sigma_z^2 \end{bmatrix} \tag{4.4}$$

As previously mentioned, the value of sensor noise is typically provided by the manufacturer. However, in this case, we do not have access to such value, so we approximated it by comparing the data reported by the sensors and the ground truth of the UAV's flight. To this end, we executed the algorithm presented in Section 3.2.4 for calculating the correspondence between the ground truth data and the data reported by the sensors.

After creating the correspondence, we computed the absolute difference between the two sets of values and then we used the `cov` function of the `Numpy` library in order to compute the matrix ***R***.

Finally, we specified the **initial conditions** for the filter. To this end, we set the initial state to the position of the Alvira sensor, since it is the first sensor to detect the drone and it does so when the UAV is close to its position. We decided to set the rest of the state variables to zero, as we don't have any information about their values. Thus, the initial state would be:

$$\boldsymbol{x}_0 = \begin{bmatrix} 5.85862734 & 51.52126391 & 31 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{4.5}$$

As part of this definition of the initial conditions of the tracking task, we decided to initialize the **covariance matrix *P*** with the identity matrix, since we did not have any reason to modify that default value.

4.3 Results

In order to evaluate the performance of this model, we decided to consider the first scenario (Scenario 1.1) as the training set and then use the remaining scenarios (Scenario 1.2, Scenario 1.3, and Scenario 1.4) as the testing sets. The reason for this decision was the fact that Scenario 1.1 is the simplest scenario according to Figure 3.2. Thus, we considered it would work better as a baseline and that it would be easier to deduce the behavior of the UAV in relation to the sensor data from it.

In the context of the linear Kalman Filter model, we considered the training process to consist of using the data of Scenario 1.1 to compute the values of the matrices and parameters that depend on the input measurement data (***R*** and Δt) so that we can use those computed values when evaluating the remaining scenarios.

After the training phase, we moved on to a testing phase in which we ran the Kalman Filter algorithm for every scenario (Scenario 1.1, Scenario 1.2, Scenario 1.3, and Scenario 1.4), considering all the values reported by the sensors in each particular scenario.

We chose this methodology for testing because we believed that it would more closely resemble the application of this algorithm in an online scenario in which all the incoming data should be considered since we cannot make any *a priori* filtering of the values reported by the sensors.

Scenario	Coordinate	MAE	MSE	NRMSE	NCIAMetric	Time
Scenario 1.1	Longitude	0.00919	0.00012	0.00188	15.12216	1.31s
	Latitude	0.00361	0.00002	0.00009		
	Altitude	20.62431	650.73599	0.39937		
Scenario 1.2	Longitude	0.01279	0.00026	0.00275	15.39332	10.42s
	Latitude	0.00475	0.00004	0.00012		
	Altitude	20.56227	664.49864	0.37514		
Scenario 1.3	Longitude	0.00902	0.00013	0.00192	18.80884	3.30s
	Latitude	0.00361	0.00002	0.00009		
	Altitude	26.04831	883.81749	0.65799		
Scenario 1.4	Longitude	0.01202	0.00021	0.00248	20.85696	6.38s
	Latitude	0.00304	0.00002	0.00008		
	Altitude	28.57824	1223.03196	0.55660		

Table 4.1: Results Kalman Filter

However, as a result of this decision, we obtained many more predictions than the number of ground truth values we had in order to evaluate the prediction process. The way we addressed this issue was to apply once again the algorithm for finding a correspondence between the predictions and the ground truth according to their timestamps as defined in Section 3.2.4.

It should be noted that for the training phase, and in particular for the computation of the $\boldsymbol{R}$ matrix, the algorithm shown in 3.2.4 defines a correspondence between the ground truth data and the sensor data of Scenario 1.1, whereas for the testing phase of Scenario 1.1 the correspondence is between the ground truth of Scenario 1.1 and the predictions. There is an important distinction between those two correspondences because the timestamps of the predictions are different from the timestamps of the sensor data. The reason for this is that the prediction process generates a prediction for every Δt timestep according to the dynamic model, which therefore differs from the irregular timestep found in the timestamps of the sensor data. Thus, this difference between the temporal correspondence in the training set and the testing set of Scenario 1.1 would likely explain that some of the error metrics in Table 4.1 return higher values for Scenario 1.1 than for the other scenarios.

Finally, according to the described evaluation strategy, the results obtained with the linear Kalman Filter model are shown in Table 4.1. The first observation that we can make about these results is that while the errors for the latitude and longitude are quite low for all the scenarios, the altitude has a significantly larger error across all metrics. This may be explained by the lack of precision that the radar sensors have when measuring that dimension, as mentioned previously. As for the NCIAMetric, we can observe that the errors obtained are very similar to those reported by the third and fourth teams in the competition, as mentioned previously.

Moreover, Table 4.1 shows the time required for the initialization of the model and the prediction in each scenario. It should be mentioned that the value of these reported times is dominated by the prediction process, since the initialization of the model is of the order of 0.01 seconds.

As for the computation times, we also measured the time required for training (computing $\boldsymbol{R}$ and Δt), which was 2.14 seconds.

All in all, we believe that the linear Kalman Filter model using a combination of data from the two radar sensors is able to generate a good approximation of the trajectory of the UAV, producing a low error in all the studied scenarios.

CHAPTER 5

Prediction model with Deep Learning

Deep Learning is a subfield of Machine Learning that puts the focus on algorithms inspired by the structure and behaviour of the brain, called artificial neural networks. The reason that motivated us to use Deep Learning models for predicting the trajectory of a UAV is that they are a black-box approach that contrasts with the previously considered white-box and transparent approach of the Kalman Filter algorithm. In this context, Deep Learning learns to approximate the hidden function that governs the behaviour of the UAV instead of requiring us to directly model the parameters that define the learning of the underlying dynamics of the system.

This chapter is structured as follows. Section 5.1 provides a theoretical overview of the Deep Learning models considered in this project. Section 5.2 defines the way in which we formulated the problem and how we designed the models in order to obtain experimental results. Finally, Section 5.3 shows the results (and the corresponding analysis) obtained by applying the Deep Learning models to the data obtained from the NCIA dataset [1].

5.1 Theoretical overview of Neural Network-based architectures

In this project, we decided to focus on two different Deep Learning models: FeedForward Neural Networks (FFNN) and Long Short-Term Memory (LSTM).

On the one hand, we considered the FFNN model because it is the simplest Deep Learning model and yet capable of approximating very complex functions. Our intuition is that the FFNN model could show good performance as well as not require a very complex and costly process for its implementation and training.

On the other hand, Recurrent Neural Networks (RNNs) in general are especially suited for working with the type of sequential data we handle in this project, and LSTM, in particular, is a commonly adopted model, since it avoids the long-term dependency problem present in vanilla RNNs.

5.1.1. FeedForward Neural Networks

FeedForward Neural Networks (FFNN) are artificial neural networks in which the connections between the nodes do not form a cycle. These models receive the name feedforward because the information is only processed in the forward direction, moving from one layer to the next one.

The simplest form of a FFNN is the single-layer Perceptron. This model computes a weighted sum of the values in the input layer and produces an activated value (typi-

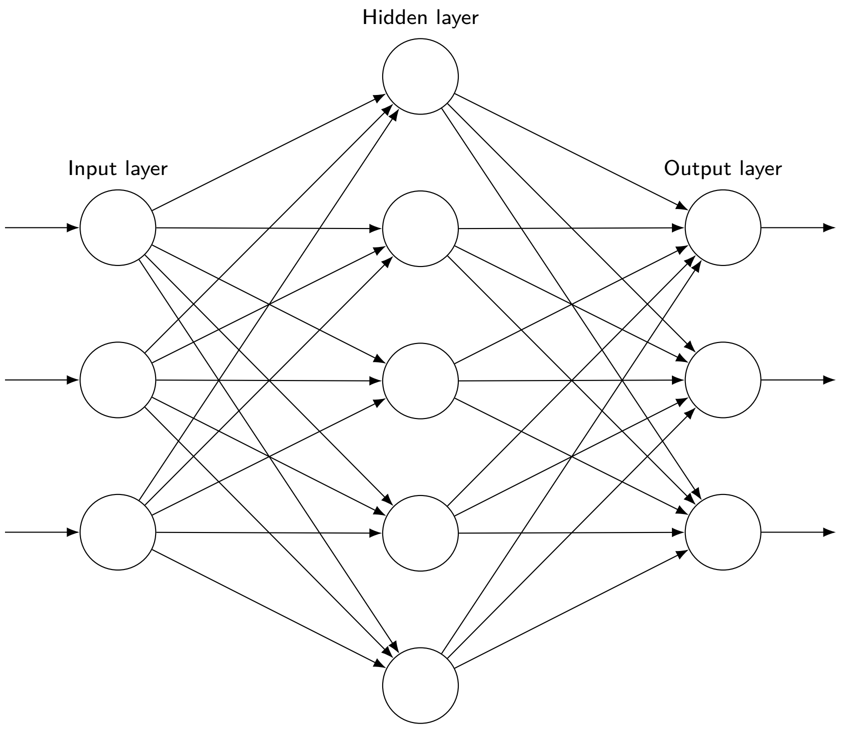


Figure 5.1: Diagram of a FFNN

cally 1) when the sum is above a certain threshold (typically 0) and a deactivated value (typically -1) otherwise.

The property known as *the delta rule* allows to compare the outputs of the model with the intended values, therefore allowing the network to adjust its weights through a training process toward producing more accurate output values. The learning process of the network is then an implementation of a form of gradient descent.

Even though the single-layer Perceptron is a very interesting model, it has the limitation of only learning linearly separable patterns.

When trying to approximate more complex functions, we need to consider models of greater complexity. In particular, the Multi-layer Perceptron (also called Feedforward Neural Networks) has at least three layers, each containing a variable number of nodes: input layer, hidden layer (one or more), and output layer. This model is represented in Figure 5.1.

Except for the input nodes, at least one node in the other layers must have a non-linear activation function because otherwise, the model would compute a linear function, being then equivalent to the single-layer Perceptron. Moreover, a Multi-layer Perceptron is trained using a process called backpropagation, in which the weights of the network are adjusted according to the comparison of the outputs and the intended values according to some error function in an application of gradient descent.

5.1.2. Long Short Term Memory

Long Short-Term Memory (LSTM) networks are an advanced version of RNNs, and as such, they present a similar structure. LSTM networks consist of multiple layers just like a vanilla FeedForward Neural Network, but with the important difference of containing recurrent units in its hidden layer. These units allow the processing of sequential data by

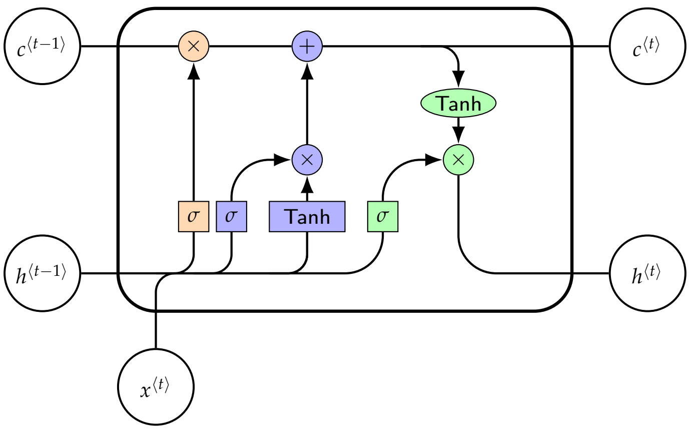


Figure 5.2: Diagram of an LSTM cell

recurrently passing a hidden state from a previous time step and combining it with the input of the current one.

In particular, LSTM networks were designed as a way to overcome the long-term dependency problem present in RNN due to the vanishing gradient problem. This problem occurs when training artificial neural networks with gradient-based learning methods and backpropagation. When using these types of methods, the weights of the network are updated at each training iteration proportionally to the partial derivative of the error function with respect to the current weight.

In some cases, it can occur that the gradient will be vanishingly small, and thus it would prevent the weight from changing value. This can sometimes result in an interruption of the network's training process. LSTM networks address this problem by incorporating a cell state whose information is controlled by three *gates*:

1. **Forget gate**: This gate decides which bits of the cell state should be maintained taking into account the previous hidden state and the new input data.

2. **Input gate:** This gate aims to determine what new information should be added to the cell state given the previous hidden state and the new input data.

3. **Output gate:** This gate decides the new hidden state by considering the newly updated cell state, the previous hidden state, and the new input data.

The diagram of Figure 5.2 shows the structure of an LSTM cell, including the three mentioned gates as well as the different operations and functions that are applied to the data:

- $c^{\langle t-1\rangle}$ is the previous cell state
- $h^{\langle t-1\rangle}$ is the previous hidden state
- $x^{\langle t\rangle}$ is the new input data
- $c^{\langle t\rangle}$ is the new cell state
- $h^{\langle t\rangle}$ is the new hidden state
- σ represents the sigmoid function
- Tanh represents the hyperbolic tangent function

- $+$ represents the pointwise addition
- $\times$ represents the pointwise product

, and the colors identify the components of the gates according to the following correspondence:

- 🟧 : Forget gate
- 🟦 : Input gate
- 🟩 : Output gate

5.2 Problem formulation with Deep Learning models

In this project, we used the Python library `Keras` to define both the FFNN and the LSTM models. This library is a high-level API that allows for a more comfortable and easy-to-use experience when interacting with backend neural network computation libraries, such as Tensorflow or Theano. We opted for working with Tensorflow, as it is the most commonly used and well-known option.

In order to implement the LSTM network with `Keras`, we defined a `Sequential` model with three layers: input layer, LSTM layer, and output layer. The input layer serves as a way to provide the input data to the LSTM layer, and the output layer is in charge of transforming the output of the LSTM layer into the expected output of the network. Similarly, in the case of the FFNN, we also defined a `Sequential` model with three layers, but replaced the LSTM layer with a Dense layer.

The input for both models is the data reported by the sensors, which consists of the values of the three observed variables: longitude, latitude, and altitude. Similarly, we want the network to provide three values at each time step, corresponding to the same three coordinates, which will serve as the prediction of the network. Therefore, both the input and the output layers will consist of three units.

In order to optimize the performance of both the LSTM and the FFNN model, we decided to explore multiple values for various hyperparameters of the models. The first hyperparameter we studied was the **number of units of the Dense and LSTM layer**, which is a positive number that defines the dimensionality of the output space of the layer. In this project, we decided to consider the values $\{10, 20, 30, 40\}$.

We also studied the performance of the models for **different activation functions** of the LSTM and Dense layer. In particular, we tested the ReLU activation function and the tanh function (hyperbolic tangent function). After running some initial experiments, we observed that the tanh function clearly outperformed ReLU, so we decided to use the tanh activation function.

Also, we analyzed **different optimizers** and different values for their **learning rate**. Specifically, we considered the Adam and Stochastic Gradient Descent (SGD) optimizers and the values $\{0.01, 0.05, 0.1, 0.5\}$ for the learning rate.

Furthermore, we also decided to explore some **hyperparameters** related to the training process, such as the number of epochs and the batch size. In that sense, we considered the values $\{100, 200, 500, 700\}$ for the number of epochs, and the values $\{1, 8, 16, 32, 64\}$, for the batch size.

Hyperparameter	**Value**
Units	20
Optimizer	SGD
Learning rate	0.1
Epochs	500
Batch size	32

Table 5.1: Optimal combination of values for the hyperparameters of the FFNN model

Hyperparameter	**Value**
Units	30
Optimizer	SGD
Learning rate	0.1
Epochs	500
Batch size	16

Table 5.2: Optimal combination of values for the hyperparameters of the LSTM model

Another important consideration is the **loss function** of the model. In our case, we decided to utilize the Mean Square Error, since it is one of the metrics we are considering to analyze the performance of the models.

In order to determine the best combination of hyperparameters that optimizes the performance of these models, we ran a GridSearch process using cross-validation. For this process, we utilized the `GridSearchCV` class of the `scikit-learn` Python library. After specifying the parameter grid using the previously mentioned values for each of the considered hyperparameters, we ran the GridSearch process using 3-fold cross-validation for estimating the performance of each combination of values. The data provided to the GridSearch process is the result of applying the correspondence algorithm shown in Section 3.2.4 that matches the ground truth and the sensor data of Scenario 1.1.

The resulting optimal combinations of values of the hyperparameters for the FFNN and LSTM models are respectively shown in Tables 5.1 and 5.2.

It should be mentioned, however, that both the number of epochs and, more specifically, the batch size have a very significant impact on the computation time required for training the network. It is then important to take into account that even though the values we have selected are the ones that empirically provide the best performance, they are not the ones that guarantee the fastest training process. In this sense, selecting a lower number of epochs or a higher value for the batch size would drastically decrease the time required for training, but a trade-off would occur insofar as the errors would be higher.

This particular trade-off situation would require careful consideration in the case of applications of this model in an environment that has strict time requirements for training the models. However, for this project we decided to limit the scope of this analysis to determining the model with the lowest errors, since we are not subject to time requirements.

Scenario	Coordinate	MAE	MSE	NRMSE	NCIAMetric	Time
Scenario 1.1	Longitude	0.00409	0.00002	0.00077	8.96771	1.22s
	Latitude	0.00148	0.00000	0.00003		
	Altitude	12.14222	357.95352	0.31746		
Scenario 1.2	Longitude	0.00980	0.00094	0.00524	5.13965	8.63s
	Latitude	0.00788	0.13029	0.00701		
	Altitude	3.79385	50.78177	0.10365		
Scenario 1.3	Longitude	0.00612	0.00005	0.00124	16.60933	2.47s
	Latitude	0.00264	0.00001	0.00006		
	Altitude	23.15924	549.25490	0.51871		
Scenario 1.4	Longitude	0.00693	0.00253	0.00859	13.48905	4.98s
	Latitude	0.01061	0.38062	0.01198		
	Altitude	15.98248	308.30760	0.27946		

Table 5.3: Results FFNN

5.3 Results

In order to obtain the results of applying the error metrics to the previously explained FFNN and LSTM models, we followed a similar strategy to the one exposed in the Results section of the Kalman Filters chapter.

We used the data of Scenario 1.1 to train the network and then tested the resulting model in all the scenarios. The reason for choosing Scenario 1.1 for training is because it is the simplest one in terms of the trajectory followed by the UAV. In turn, we expect that generalizing a prediction mode will be easier from this scenario.

The training process of the FFNN and LSTM models requires a labeled training set; that is, for each input value, we must associate the expected output value. However, as said before, there are many more sensor data samples than the number of ground truth data in the log files. For this reason, once again, we applied the algorithm defined in Section 3.2.4 to establish a correspondence between the sensor data and ground truth data of Scenario 1.1 according to the timestamp of the samples. As a result of this process, for each input data of the combined sensor data, we obtained the corresponding label as provided by the ground truth data.

After the training phase, we obtained the FFNN and LSTM models that will be used in the prediction of each scenario. In order to carry out the prediction process, we fed all the sensor data values of each scenario to the model, receiving the corresponding predictions as the output of the network. Afterwards, we apply once more the algorithm shown in Section 3.2.4 in order to match the predictions with the corresponding ground truth values so as to compute the error.

The resulting values for the error metrics of FFNN and LSTM are shown respectively in Figure 5.3 and Figure 5.4.

The first observation about these results is that the errors for the latitude and longitude are again quite low for all the scenarios, whereas the altitude has a significantly larger error across all metrics. This may be explained by the lack of precision that the radar sensors when measuring the altitude dimension, as mentioned previously. As for the NCIAMetric, we can observe that the errors obtained are very similar to those reported by the top three teams in the competition. Furthermore, we can see that according

Scenario	Coordinate	MAE	MSE	NRMSE	NCIAMetric	Time
Scenario 1.1	Longitude	0.00401	0.00002	0.00076	9.68119	1.43s
	Latitude	0.00146	0.00000	0.00003		
	Altitude	13.26497	313.83150	0.29725		
Scenario 1.2	Longitude	0.00945	0.00042	0.00349	5.77129	7.37s
	Latitude	0.00470	0.00639	0.00155		
	Altitude	6.41131	62.39446	0.11489		
Scenario 1.3	Longitude	0.00584	0.00005	0.00118	14.47623	2.18s
	Latitude	0.00255	0.00001	0.00006		
	Altitude	20.10275	417.02477	0.45198		
Scenario 1.4	Longitude	0.00629	0.00084	0.00496	11.76167	4.44s
	Latitude	0.00467	0.03209	0.00348		
	Altitude	15.32373	283.21770	0.26785		

Table 5.4: Results LSTM

to the NCIAMetric, the FFNN model performs better in Scenario 1.1 and Scenario 1.2, while LSTM performs better in Scenario 1.3 and Scenario 1.4.

Tables 5.3 and 5.4 also show the time required for the prediction in each scenario. We can see that the fastest model is dependent on the scenario. Nevertheless, the reported times of both the FFNN and the LSTM models are very close in all scenarios, so there is no significant difference in that regard. However, the training time required by the FFNN model was 164.06 seconds, while the time required by LSTM was 438.99 seconds. We can thus conclude that the simpler FFNN model required almost a third of the time of LSTM to be trained.

All in all, we believe that both the FFNN and LSTM are able to generate a good approximation of the trajectory of the UAV, producing a low error in all the studied scenarios.

Finally, we would like to draw the reader's attention to the high error values of Scenario 1.1 compared to those of Scenario 1.2. Here the potential mismatch between the temporal correspondence in the training set and the testing set of Scenario 1.1 is deepened further than in the case of the Kalman Filter algorithm. This situation reveals that when the training data distribution is not a faithful representation of the test data distribution, Neural Network-based models are affected to a greater extent than a non-neural model.

CHAPTER 6
Performance comparison

In this chapter, we will compare the performance of the Kalman Filter algorithm and the Deep Learning models explained in Chapters 4 and 5, respectively.

With the aim to provide a comprehensive analysis of the behaviour of these models, we will first compare the errors obtained with the different models in Section 6.1 and then compare the time required by the training and prediction phases of the models in Section 6.2. Finally, Section 6.3 contains the conclusions we derived from the previous comparison and analysis of both the errors and the computation time of the models.

6.1 Error comparison

In this analysis, we will only consider the NRMSE and NCIAMetric metrics to compare the performance of the different models. The reason for not including MAE and MSE in this analysis is because NRMSE expresses similar information to these two metrics, and besides, NRMSE is easier to interpret (values between 0 and 1). As for NCIAMetric, it provides additional information since it integrates the errors of all dimensions (longitude, latitude, and altitude).

Figures 6.1 and 6.2 show the NRMSE of the longitude and latitude for each scenario according to the results obtained from the testing process of each model. We will analyze both of those figures simultaneously because we can observe very similar patterns, although slightly accentuated in the case of the latitude. In that sense, we can observe two different patterns in those figures, one present in Scenario 1.1 and Scenario 1.3 and a different one present in Scenario 1.2 and Scenario 1.4. On the one hand, Scenarios 1.1 and 1.3 show that both of the Deep Learning models have similar errors and that they outperform the Kalman Filter algorithm. On the other hand, Scenarios 1.2 and 1.4 show that the Kalman Filter outperforms the Deep Learning models, and also that FFNN has a higher error than LSTM. However, it should be mentioned that even though the difference in the errors may appear significant, the errors reported in those figures are all extremely low, as can be observed from the values shown on the y-axis. Thus, all the models provide extremely good predictions for both the longitude and the latitude dimensions of the UAV flight. Another observation we can make about these figures is that the Kalman Filter provides a more robust performance across all scenarios, while the Deep Learning models have both very low or very high errors depending on the scenario.

Figure 6.3 shows the NRMSE of the altitude for each scenario according to the results obtained from the testing process of the models. We can observe that the error of the Kalman Filter is higher than the error of the Deep Learning models in all scenarios. Moreover, FFNN and LSTM have similar errors, but it should be mentioned that LSTM

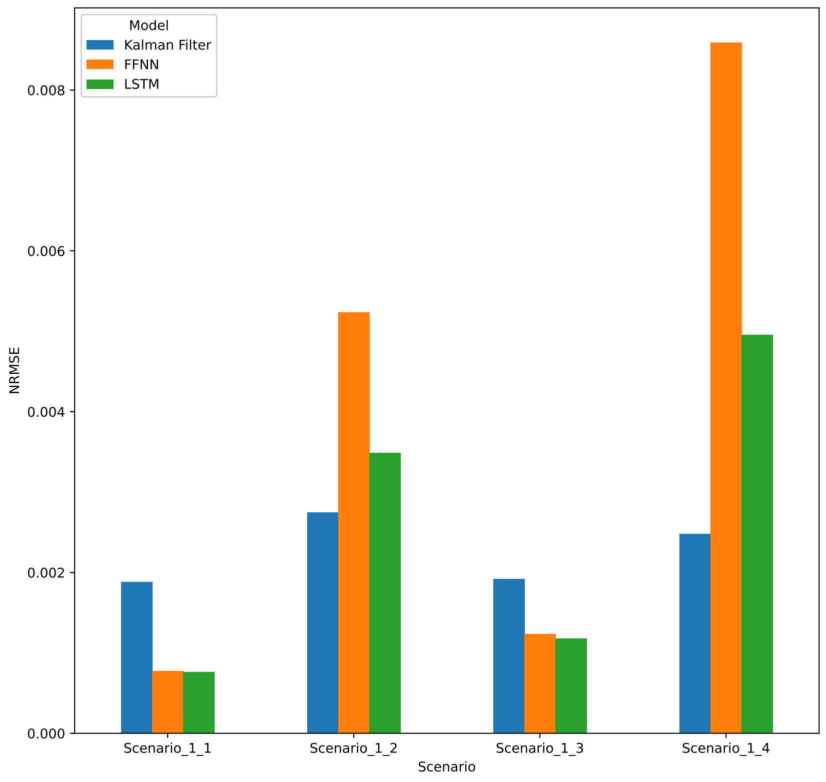


Figure 6.1: NRMSE of the longitude for each scenario resulting from the testing process of each model

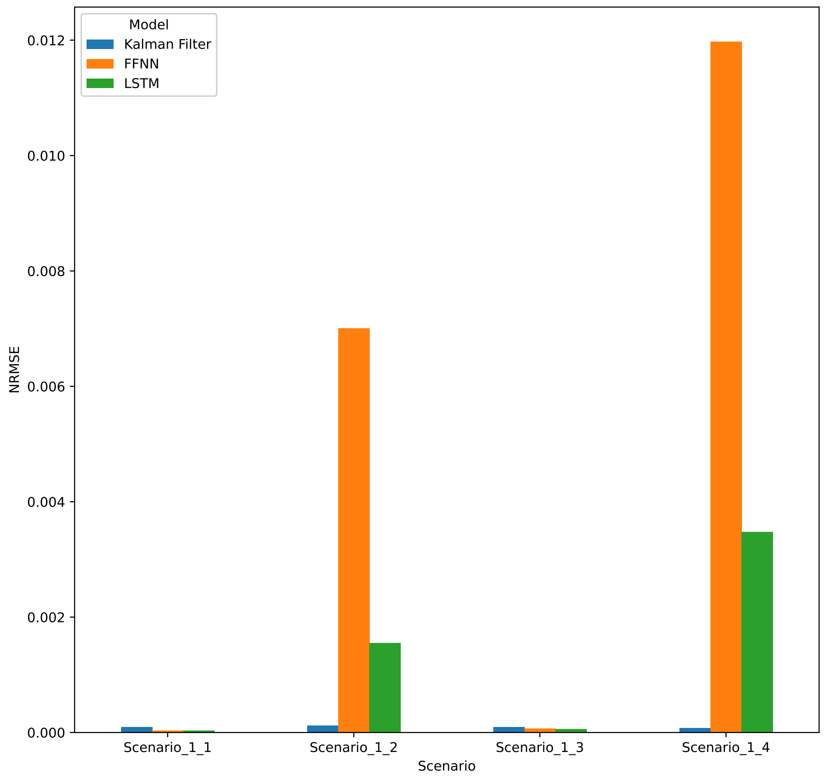


Figure 6.2: NRMSE of the latitude for each scenario resulting from the testing process of each model

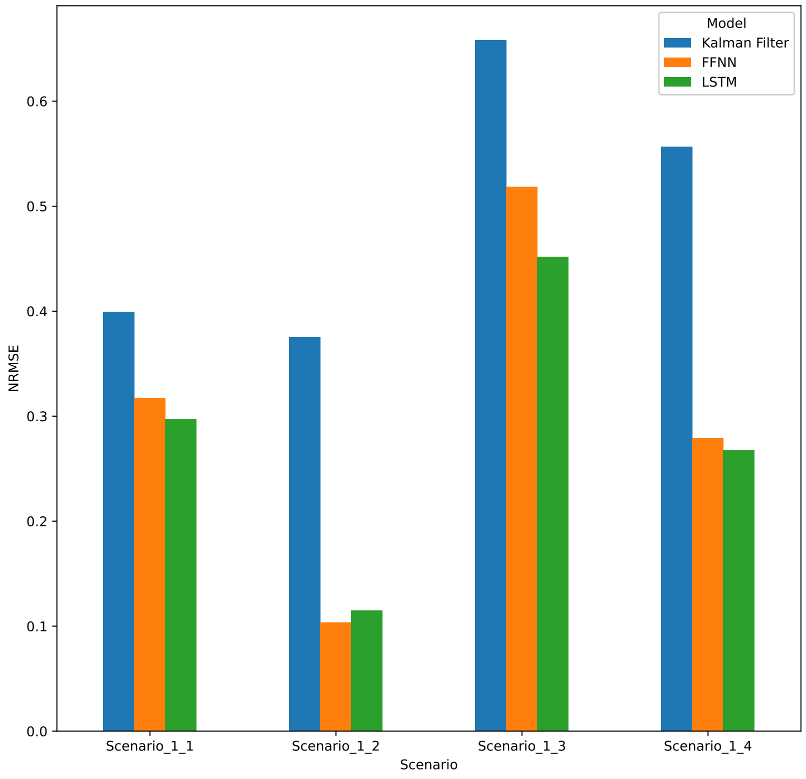


Figure 6.3: NRMSE of the altitude for each scenario resulting from the testing process of each model

slightly outperforms FFNN in all scenarios except for Scenario 1.2. Another observation is that conversely to the longitude and latitude errors, in this case, the errors are all quite high (close to 1) in all the scenarios and for all the models. This illustrates the intrinsic difficulty in predicting the altitude dimension of the UAV flight, which, as we previously mentioned, is probably due to the lack of precision of the radar sensors when measuring such dimension.

Lastly, Figure 6.4 shows the error obtained from the NCIAMetric metric for each scenario according to the results obtained from the testing process of the models. We can observe that, once again, the Kalman Filter is the model that generates the highest error in all the scenarios. Moreover, we can see that FFNN outperforms LSTM in Scenarios 1.1 and 1.2, whereas LSTM provides a lower error for Scenarios 1.3 and 1.4. Another observation that we can make about this figure is that all of the reported errors are lower or equal to the errors reported by the top four teams in the competition [1]. Nevertheless, we should recall again that the competition used unknown tests for assessing the performance of the proposed solutions and thus, the errors we obtained are not completely comparable to those reported in the competition.

6.2 Computation time comparison

After analyzing the performance of the models according to their errors in the testing, we will now focus on comparing the time required by the models in the training and prediction phases.

Figure 6.5 shows the time each model required for training. As we can see, the Kalman Filter needs a considerably lower amount of time to be trained compared to the Deep Learning models. This difference is due to the fact that the Kalman Filter does not

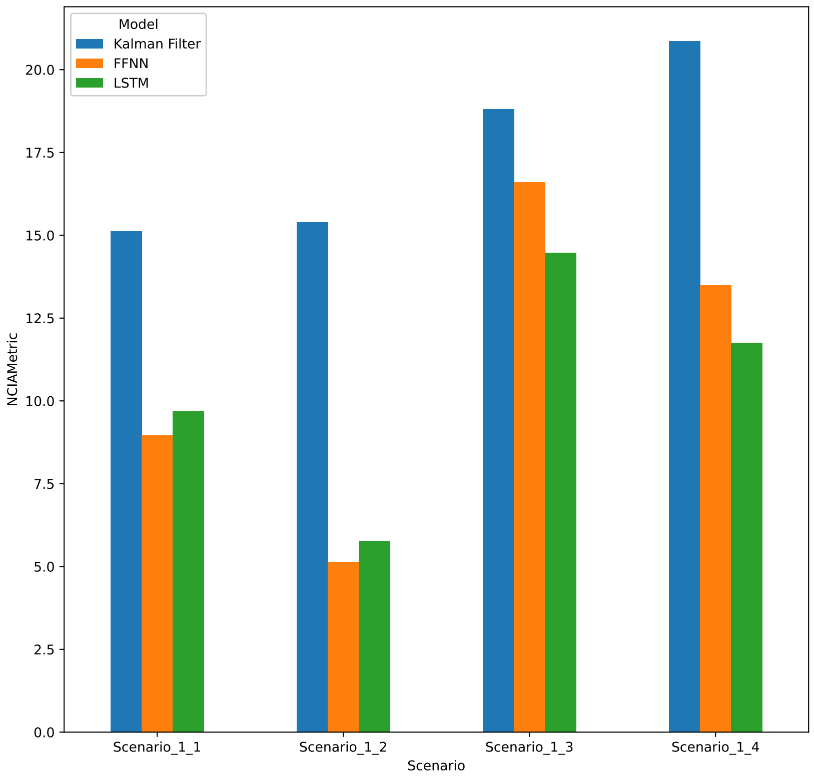


Figure 6.4: NCIAMetric error for each scenario resulting from the testing process of each model

need to learn the parameters of the model, since we manually specify their values. Consequently, we considered the training phase of the Kalman Filter to only consist of the computation of the $\boldsymbol{R}$ matrix and the time step Δt, which are both very straightforward and quick-to-make calculations. Conversely, the Deep Learning models do need to learn the parameters of the models according to the training data, and this process requires much more time. Moreover, FFNN requires much lower time than LSTM because it is a much simpler model, and thus there are far fewer parameters that need to be optimized in the training phase.

Figure 6.6 shows the time each model required for generating the predictions in the testing process of each scenario. We can observe that the Kalman Filter is the model that requires more time to generate the predictions in all scenarios except for Scenario 1.1. Conversely, LSTM requires less time to generate the predictions in all scenarios except for Scenario 1.1. It should also be mentioned that Scenario 1.2 is the most time-consuming in all models because it contains the largest amount of sensor data samples, and so more time is required in order to process them and generate the corresponding predictions.

6.3 Concluding remarks

After comparing and analyzing the previously discussed errors and computation times, we aim to draw some general conclusions about the overall performance and behaviour of the models.

Firstly, we will point out some of the main differences between the Kalman Filter and the Deep Learning models in order to properly frame the subsequent comparison. One first important difference lies in that the Kalman Filter algorithm approximates a linear model, whereas the Deep Learning models are capable of approximating a non-linear

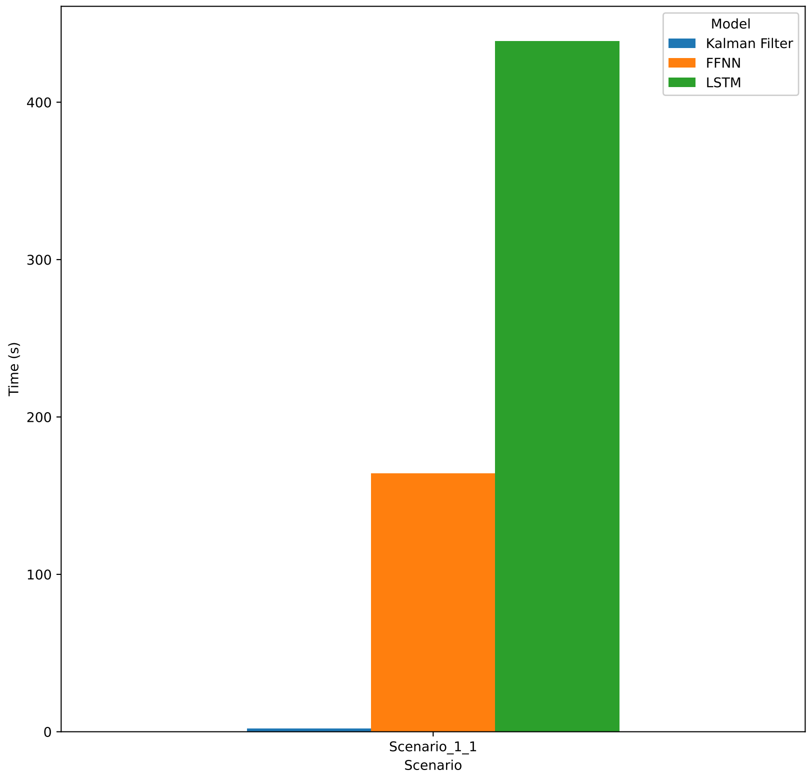


Figure 6.5: Time required by the training process of each model

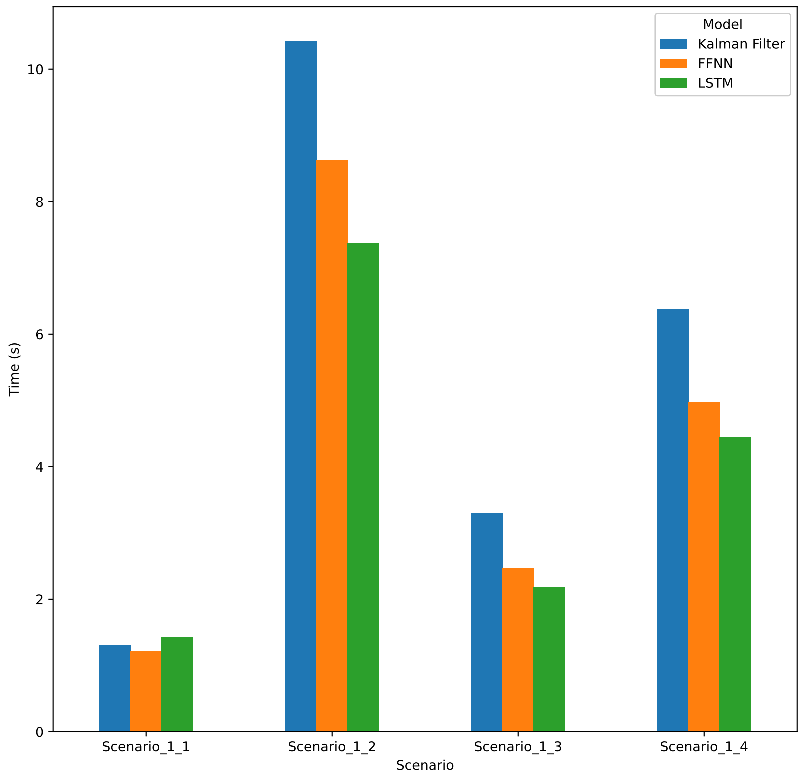


Figure 6.6: Time required by the prediction process of each model in each scenario

model. This means that the Deep Learning models are more flexible for accurately approximating certain irregular patterns in the data. Another difference is that we manually specified the values of the model parameters in the Kalman Filter algorithm, while the Deep Learning models learn the optimal values of the parameters of the model according to the training data. This difference has two main consequences. The first one is that the behaviour of the Kalman Filter model is easier to understand and explain. The second one is that the values of the parameters of the Deep Learning models have been optimized through the training process, while this cannot be guaranteed in the Kalman Filter approach.

The main observation we can make about the obtained results is that all of the models are able to provide a very good approximation for the longitude and latitude of the UAV, and that the main source of error for all the models is the altitude dimension.

On the one hand, as we can see in the errors of the Deep Learning models, there is a highly intrinsic difficulty in predicting that dimension even when using a non-linear model. That difficulty may be partly alleviated using more complex Deep Learning models, but the results do not show a clear way in which the current models may be improved due to the black-box nature of these models.

On the other hand, the higher errors obtained with the Kalman Filter lead us to conclude that both the chosen dynamic model and the specified values for the parameters may have room for improvement. In particular, we believe that the approach we followed to approximate the Measurement Noise Matrix $\boldsymbol{R}$ may not be able to completely capture the noise present in the altitude dimension of the data reported by the sensors. In this sense, having access to the sensor noise reported by the manufacturer or to a more precise noise approximation might have led to better results. In addition, even though the non-constant acceleration dynamic model seems to properly approximate the behavior of the UAV, other alternatives might make better adjustments to its behaviour. We believe this avenue could also be explored in order to seek an improvement in the performance of the Kalman Filter model.

All in all, the main conclusion that we can draw from this analysis is that while the Deep Learning models are able to produce more accurate predictions, the more explainable nature of the Kalman Filter model allows us to find easier interpretations of the UAV behaviour and thus of the different approaches that could be investigated in order to improve its performance.

CHAPTER 7
Conclusions

This chapter serves as a conclusion to the previously described work that was carried out for this project. In that sense, we will first analyze whether we were able to fulfill all the objectives we proposed for this project in Section 7.1. Then, Section 7.2 describes the most relevant relations between the work carried out in this project and the courses of the Bachelor's Degree in Informatics Engineering. Finally, Section 7.3 specifies some of the main research directions that could be explored in order to expand the work presented in this project.

7.1 Fulfilment of the project's objectives

We believe that, as a consequence of all the work presented in this project, we were able to fulfill all the objectives outlined at its start in Chapter 1. In that sense, in the following list, we specify the part of this project in which we fulfilled each of those objectives.

1. We were able to identify the main techniques and approaches related to the task of tracking a UAV, which we described in Chapter 2.
2. In Chapter 3, we studied and analyzed the sensor data contained in the publicly available dataset NCIA dataset, and we carried out the necessary preprocessing steps in order to prepare the data for its usage by the models.
3. Later, in Chapter 4, we presented the main theoretical characteristics of a linear Kalman Filter model, and we studied its performance for the problem of tracking a UAV.
4. Then, in Chapter 5, we exposed some of the main theoretical characteristics of the Deep Learning models we considered, and we analyzed their performance for the problem of tracking a UAV.
5. Finally, in Chapter 6, we performed a detailed comparison of the performance of the considered Kalman Filter and Deep Learning models, and we extracted some conclusions about their behavior in this specific problem.

7.2 Relation with courses of the Bachelor's Degree

The knowledge and skills acquired in many of the different courses in the Bachelor's Degree in Informatics Engineering have been very helpful for developing this project.

Several courses related to Mathematics, such as *Algebra*, *Discrete Mathematics*, *Statistics*, and *Mathematical Analysis*, were extremely useful when exploring and understanding the behaviour of the Kalman Filter model.

Moreover, the courses related to Artificial Intelligence, and particularly to Machine Learning, were instrumental in understanding and applying the Deep Learning models.

Furthermore, other courses such as *Information storage and retrieval systems* and *Algorithmics* were useful insofar as the usage of the programming language `Python` in their practical sessions allowed for familiarity with the usage of the language in the context of a project prior to the implementations that were necessary for this work.

7.3 Future research

Due to the extensive and complex nature of the problem of tracking UAVs, our work may be expanded in many different ways. In particular, we have identified three main directions that could be followed in future research:

- **Data considered:**
  - Different types of UAVs have different flight characteristics, and thus the best model to track them may differ.
  - Consider scenarios containing multiple UAVs. As described in Chapter 3, the NCIA dataset contains several scenarios where multiple UAVs flew simultaneously. Developing a prediction model for those scenarios has the additional complexity of having to discern what part of the data reported by the sensors corresponds to each UAV.
- **Additional tasks:**
  - The dataset utilized in this project contained data of a UAV obtained in a controlled flight zone. However, in a real-life scenario, the target detected by a sensor may not necessarily be a UAV (it could be a bird, for instance). For that reason, it may be interesting to develop a method that is capable of classifying the detected target into UAV and non-UAV categories before performing the tracking.
  - Similarly, the behavior of different models of UAVs may not be identical, and thus it could be interesting to explore ways in which to incorporate an identification method so as to provide more accurate tracking performance for each specific model of UAV detected by the sensors.
- **Complexity of the models:**
  - The considered linear Kalman Filter model is one of the most robust methods for tracking a target. However, there exist many variants that attempt to improve its performance by introducing the capacity to model non-linearities in the data. Some of these are the Extended Kalman Filter (EKF) and the Unscented Kalman Filter (UKF). The additional performance that these models may provide comes at a price, however, since their parameters are generally harder to define and their behavior is less stable.
  - In this project, we considered the FFNN and LSTM Deep Learning models, since they are quite straightforward to define and they provide good performance in this particular problem. Nonetheless, it may be interesting to explore

other Deep Learning models, such as the Attention Mechanism, in order to observe whether their additional complexity is translated into better performance.

Bibliography

[1] Community Prediction Competition. Drone identification and tracking. `https://www.kaggle.com/c/icmcis-drone-tracking/`. Accessed: 2022-04-20.

[2] Alex Becker. Kalman Filter Tutorial. `https://www.kalmanfilter.net/default.aspx`. Accessed: 2022-06-17.

[3] Min Xue. UAV Trajectory Modeling Using Neural Networks. In *17th AIAA Aviation Technology, Integration, and Operations Conference*, 06 2017.

[4] Ying-Chih Lai and Zong-Ying Huang. Detection of a Moving UAV Based on Deep Learning-Based Distance Estimation. *Remote Sensing*, 12(18), 2020. URL: `https://www.mdpi.com/2072-4292/12/18/3035`, `doi:10.3390/rs12183035`.

[5] 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. `doi:10.1109/ICBASE53849.2021.00089`.

[6] Nachiket Deo and Mohan Trivedi. Multi-Modal Trajectory Prediction of Surrounding Vehicles with Maneuver based LSTMs. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1179–1184, 06 2018. `doi:10.1109/IVS.2018.8500493`.

[7] Vince Kurtz and Hai Lin. Toward Verifiable Real-Time Obstacle Motion Prediction for Dynamic Collision Avoidance. *2019 American Control Conference (ACC)*, pages 2633–2638, 2019.

[8] Ke Xiao, Jianyu Zhao, Yunhua He, and Shui Yu. Trajectory prediction of uav in smart city using recurrent neural networks. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pages 1–6, 2019. `doi:10.1109/ICC.2019.8761110`.

[9] Mohammed Al-Absi, Rui Fu, Kihwan Kim, Young-Sil Lee, Ahmed Al-Absi, and Sanggon Lee. Tracking Unmanned Aerial Vehicles Based on the Kalman Filter Considering Uncertainty and Error Aware. *Electronics*, 10:3067, 12 2021.

[10] Yuan Wei, Tao Hong, and Michel Kadoch. Improved Kalman Filter Variants for UAV Tracking with Radar Motion Models. *Electronics*, 9:768, 05 2020.

[11] Divy Raval, Emily Hunter, Sinclair Hudson, Anthony Damini, and Bhashyam Balaji. Convolutional Neural Networks for Classification of Drones Using Radars. *Drones*, 5(4), 2021. URL: `https://www.mdpi.com/2504-446X/5/4/149`.

[12] Duygu Yumu, Tevfik Sarikaya, M. Efe, Gökhan Soysal, and Thiagalingam Kirubarajan. Track Based UAV Classification Using Surveillance Radars. 02 2020.

[13] Sunxiangyu Liu, Guitao Li, Yafeng Zhan, and Peng Gao. Musak: A multi-scale space kinematic method for drone detection. *Remote Sensing*, 14(6):1434, 2022.

[14] International Conference on Military Communications and Information Systems. ICMCIS Data Challenge. https://www.icmcis.eu/challenge/, Accessed: 2022-04-20.

[15] Roger R. Labbe. FilterPy. https://filterpy.readthedocs.io/en/latest/index.html#, Accessed: 2022-06-01.

[16] Viviano S Medeiros, Renno ED Vale, Yuri C Gouveia, Wellton T Souza, and Alisson V Brito. An independent control system for testing and analysis of uavs in indoor environments. In *2016 XIII Latin American Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR)*, pages 55–60. IEEE, 2016.

[17] United Nations Development Programme. What are the Sustainable Development Goals? https://www.undp.org/sustainable-development-goals. Accessed: 2022-06-20.

[18] Airobotics. Automated Drone Applications for Industrial Purposes. https://www.airoboticsdrones.com/applications/. Accessed: 2022-06-20.

[19] Jon Walker. Industrial Uses of Drones 5 Current Business Application. https://emerj.com/ai-sector-overviews/industrial-uses-of-drones-applications/. Accessed: 2022-06-20.

[20] Acciona. Drones, the great ally of sustainability. https://www.activesustainability.com/sustainable-development/drones-the-great-ally-of-sustainability/?_adin=02021864894. Accessed: 2022-06-20.

[21] Moayad Aloqaily, Ouns Bouachir, Ismaeel Al Ridhawi, and Anthony Tzes. An adaptive UAV positioning model for sustainable smart transportation. *Sustainable Cities and Society*, 78:103617, 2022.

[22] UNICEF Office of Innovation. Drones: Addressing transport, connectivity and better emergency preparedness. https://www.unicef.org/innovation/drones. Accessed: 2022-06-20.

[23] Andrew Brown. Drones vs mosquitoes: Fighting malaria in Malawi. https://www.unicef.org/stories/drones-vs-mosquitoes-fighting-malaria-malawi. Accessed: 2022-06-20.

[24] Beth Jackson. Drones in Renewable Energy: The Future is here. https://coptrz.com/drones-in-renewable-energy-the-future-is-here/. Accessed: 2022-06-20.

[25] DJI Enterprise. Saving the Planet, One Renewable Energy Drone Inspection at a Time. https://enterprise-insights.dji.com/blog/renewable-energy-inspection-drones. Accessed: 2022-06-20.

[26] TRT World. Explained: The drones that work for peace. https://www.trtworld.com/magazine/explained-the-drones-that-work-for-peace-49463. Accessed: 2022-06-20.

[27] United Nations Peacekeeping. FEATURE: Does drone technology hold promise for the UN? https://peacekeeping.un.org/en/feature-does-drone-technology-hold-promise-un. Accessed: 2022-06-20.

[28] Paul Armstrong. How drones help the environment and tackle climate change. https://www.verizon.com/about/our-company/fourth-industrial-revolution/how-drones-help-environment-and-tackle-climate-change. Accessed: 2022-06-20.

[29] Connected Places Catapult. Drone Technology, directly and indirectly, combatting climate change. https://cp.catapult.org.uk/news/drone-technology-directly-and-indirectly-combatting-climate-change/. Accessed: 2022-06-20.

[30] Zhongyu Sun, Xiaonian Wang, Zhihui Wang, Long Yang, Yichun Xie, and Yuhui Huang. UAVs as remote sensing platforms in plant ecology: review of applications and challenges. *Journal of Plant Ecology*, 14(6):1003–1023, 2021.

[31] Susana Baena, Doreen S Boyd, and Justin Moat. UAVs in pursuit of plant conservation-Real world experiences. *Ecological informatics*, 47:2–9, 2018.

APPENDIX A
Sustainable Development Goals

Degree to which the work is related to the Sustainable Development Goals (SDGs):

Sustainable Development Goals	**High**	**Medium**	**Low**	**Not applicable**
Goal 1. **No Poverty.**				X
Goal 2. **Zero Hunger.**				X
Goal 3. **Good Health and Well-being.**		X		
Goal 4. **Quality Education.**				X
Goal 5. **Gender Equality.**				X
Goal 6. **Clean Water and Sanitation.**				X
Goal 7. **Affordable and Clean Energy.**		X		
Goal 8. **Decent Work and Economic Growth.**				X
Goal 9. **Industry, Innovation and Infrastructure.**	X			
Goal 10. **Reduced Inequalities.**				X
Goal 11. **Sustainable Cities and Communities.**	X			
Goal 12. **Responsible Consumption and Production.**				X
Goal 13. **Climate Action.**			X	
Goal 14. **Life Below Water.**				X
Goal 15. **Life on Land.**			X	
Goal 16. **Peace, Justice and Strong Institutions.**		X		
Goal 17. **Partnerships for the goals.**				X

Reflection on the relationship of the TFG/TFM with the SDGs and with the most related SDG(s):

The 17 Sustainable Development Goals (SDGs) were adopted by the General Assembly of the United Nations in 2015, and they are intended to act as a layout to achieve a better and more sustainable world for all by 2030.

We have considered the main purpose behind each of those goals, as described in [17], and we have analyzed which of them are most related to the work presented in this project.

Our conclusion was that the development of accurate and reliable models to track a UAV, which we explored in this work, would have as a consequence the possibility of more deeply integrating the usage of UAVs in a larger amount of sectors, many of which are represented in various of the aforementioned SDGs.

In that sense, we believe that this work is most related to the following goals:

- **Goal 9. Industry, Innovation and Infrastructure**: The usage of UAVs has become increasingly common in several different industries, becoming a driving force for innovation. For instance, [18] showcases some ways in which UAVs can create opportunities for the development of more efficient processes in different industries, such as mining, sea ports, oil and gas. Similarly, [19] mentions five different industrial uses of UAVs: monitoring of agriculture crops, development and maintenance of infrastructures, using a drone to bring Internet access to remote areas, inspection and surveillance of construction projects, and inventory of warehouse items.
- **Goal 11. Sustainable Cities and Communities**: Nowadays, there are a lot of proposals and research works that focus on how UAVs can play a crucial role in the creation of new ways of managing smart and sustainable cities. As an example, [20] explains some of the possibilities that the usage of UAVs would enable. For instance, they could be utilized for monitoring the air quality of cities or for obtaining images of the site of an incident, thus avoiding the pollution derived from the usage of a different vehicle. Moreover, [21] proposes a solution for automatically adapting the position of a UAV in order for it to be able to support a smart transportation system by improving the communication coverage of vehicular nodes.

Other goals with which this project is strongly related are the following ones:

- **Goal 3. Good Health and Well-being**: This is another very important goal to which the usage of UAVs can contribute. For example, in [22] a wide range of applications of UAVs in the context of facing humanitarian challenges. Some of those applications are the delivery of vaccines, improving the connectivity of hard-to-reach communities, and obtaining aerial images in order to better respond to emergencies. Another interesting case of using UAVs in the context of trying to improve and guarantee good health is presented in [23]. That article describes an innovative approach for the surveillance of populations of mosquitoes in order to control the spread of malaria.
- **Goal 7. Affordable and Clean Energy**: UAVs also have the potential to enable great innovations in the field of renewable energies. For instance, [24] describes several ways in which UAVs can be utilized in the solar, wind, and wave power industries. Furthermore, [25] provides additional insights about the inclusion of UAVs in those same renewable energy industries, and it adds information about the potential usage of UAVs for the inspection of hazardous areas in nuclear power plants.

- **Goal 16. Peace, Justice and Strong Institutions**: Lastly, this is another important goal to which the usage of UAVs would greatly contribute. In [26], different ways in which UAVs can be utilized for peace and humanitarian operations are described. For instance, UAVs could work as a way of providing or guaranteeing safe transportation of life support supplies or as a way of investigating areas of conflict. Moreover, [27] describes other applications of UAVs in this context. For example, they could be used in order to provide Internet access to refugee camps or schools, and to perform other many different humanitarian tasks.

Finally, there are some other goals which also have a lower, but still considerable relation to this project:

- **Goal 13. Climate Action**: UAVs have many different applications that could help alleviate the effects of climate change. Various of those applications are described in [28]. For instance, the ability of UAVs to access remote areas allows researchers to understand how weather patterns are affecting rainforest trees, mountain air and underwater ecosystems. Moreover, other important applications are shown in [29]. Among them, UAVs can contribute to combating climate change by providing cheaper and more effective ways to document, monitor, prevent and tackle it.
- **Goal 15. Life on Land**: There are various applications of UAVs that have been proposed in order to protect plant and animal life on land. For instance, [30] describes one of these applications, in which UAVs could be used in order to carry out plant ecology research. Furthermore, other applications are mentioned in [31], such as plant conservation applications or applications for mapping, quantifying and monitoring plant species.