Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

FINAL DEGREE PROJECT

Title: Assessing drone trajectory error and improving flightpath predictability

Degree: Bachelor's degree in Aerospace Systems Engineering

AUTHOR: Pau Terés i Teixiné

DIRECTOR: Cristina Barrado Muxí

DATE: June 19th 2023

Títol: Avaluació dels errors en trajectories de drons i millorant la seva predictibilitat

Autor: Pau Terés i Teixiné

Director: Cristina Barrado Muxí

Data: 19 de juny del 2023

Resum

El creixement ràpid de la indústria dels drons té com a objectiu desenvolupar aplicacions i implementar-les en una àmplia gamma de sectors. Això inclou àrees urbanes concorregudes per a serveis com la vigilància, les entregues i monitoratge. En aquest context, és essencial tenir un excel·lent disseny de l'espai aeri. Aquest projecte es centra en l'anàlisi d'un conjunt de dades del projecte Very Large Demonstration de CORUS-XUAM, que contribueix a la missió de l'U-Space de desenvolupar un espai aeri segur, sostenible, eficient i totalment digitalitzat per a la Mobilitat Urbana Aèria integrada, que no interfereixi en les operacions actuals de l'ATM. El conjunt de dades inclou plans de vol, telemetria i prediccions de l'U-Space per a 72 vols de drons.

L'anàlisi implica comparar les trajectòries previstes amb les rutes reals per identificar els factors que contribueixen a les desviacions respecte el pla de vol i calcular els paràmetres de rendiment rellevants per evaluar l'adherència dels drons amb el pla de vol. Això es fa mitjançant l'ús d'algoritmes de deformació dinámica del temps per establir una connexió entre els punts de telemetria i el pla de vol, el que estableix la base per a la següent secció del projecte.

Després de processar les dades, en aquest projecte desenvolupem models d'aprenentatge automàtic per predir els paràmetres de telemetria basant-nos en el pla de vol proporcionat. S'avaluen diversos models per trobar el més adequat pel nostre objectiu. El projecte també implica la visualització i interpretació de les dades per obtenir una visió més intutiva del rendiment del dron i el compliment del pla de vol.

La predicció de la posició obre un nou camp de recerca i, en aquest projecte, l'enfocament és utilitzar un mètode alternatiu per definir l'espaiat i les mides de les aerovies que composen els plans de vol, de manera que es puguin establir àrees de seguretat per prevenir qualsevol possible conflicte que pugui aparèixer en futurs vols en una àrea concorreguda si l'espaiat fos inferior als llindars establerts.

Els resultats d'aquest estudi demostren una progressió exitosa des de les dades en brut fins a una anàlisi exhaustiva, que ofereix una visió valuosa per avaluar el rendiment dels drons i predir els temps de vol. El desenvolupament de diverses funcions de visualització de dades ha permès interpretar de manera eficient i efectiva les dades. Tot i que els resultats obtinguts amb el conjunt de dades disponible són notables, el potencial de millora rau principalment en l'obtenció d'un conjunt de dades més gran amb més característiques i mostres, el que milloraria el rendiment dels models d'aprenentatge automàtic i proporcionaria prediccions encara més precises.

Title: Assessing drone trajectory error and improving flightpath predictability

Author: Pau Terés i Teixiné

Director: Cristina Barrado Muxí

Date: June 19th 2023

Overview

The rapid growth of the drone industry aims to develop applications and implement them in a wide range of areas. This includes busy urban areas for services such as surveillance, deliveries and monitoring. In this context, it is essential to have an excellent design of the airspace. This thesis focuses on the analysis of a dataset from the Very Large Demonstration project of CORUS-XUAM, which contributes to the U-Space mission of developing a safe, sustainable, efficient and fully digitalized airspace for integrated Urban Air Mobility which does not interfere with current ATM operations. The dataset includes flight plans, telemetry, and U-space predictions for 72 drone flights.

The analysis involves comparing intended trajectories with actual flight paths to identify factors contributing to deviations from the flight plan and computing relevant performance parameters to assess the adherence of the drones to the flight plan. This is done with the use of dynamic time warping algorithms in order to establish a link between the telemetry points and the flight plan, which sets the basis for the next section of the project.

Having processed the data, during this project we develop machine learning models to predict telemetry parameters based on the input flight plan. Several models are tested and evaluated to find the most suitable one for our objective. The project also involves visualizing and interpreting the data to gain insights of the drone performance and adherence to the flight plan.

Position prediction opens up a new area of research and in this project the approach is to use an alternative method to define the spacing and size of the airways that compose the flight plans so as to dictate safety areas to prevent any possible conflict that could appear in future flights in a busy area if the spacing were to be below the thresholds.

The results of this study demonstrate a successful progression from raw data to a comprehensive analysis, offering valuable insights for evaluating drone performance and predicting flight times. The development of various data visualization functions enabled efficient and effective interpretation of the data. While the obtained results with the available dataset are remarkable, the potential for further improvement lies mainly in acquiring a larger dataset with more features and samples, which would enhance the performance of the machine learning models and yield even more accurate predictions.

INDEX

FIGURES

INTRODUCTION

Technology continues to advance rapidly in all aspects of our lives and this is no exception for the aeronautics industry. However, implementing new measures in this sector can be complex due to the need to consider regulations, standardization, and safety measures. A particular challenge lies in aeronautical communications, where the frequency band is saturated in busy areas, rendering it obsolete. Unfortunately, progress in resolving this issue has been slow due to the difficulties involved in making changes. Such issues should not exist in a critical and sensitive sector like aviation, which requires precision, safety, and no room for errors or mistakes.

The ability to adapt is crucial in today's industries, this implies the facts of responding to different situations, coping with changes, and mitigating threats. Aviation, which closely monitors safety procedures, adaptability is a must. As demand continues to rise and safety standards become stricter, there is a need to develop new systems that can support the future of the industry. Nearly twenty years ago, the International Civil Aviation Organization (ICAO) highlighted the expectation that air traffic would increase by 50% by 2035 compared to 2012 [1]. Upgrading current technology, including infrastructure and telecommunications, was emphasized as necessary to handle the expected high volume of air traffic.

Furthermore, the drone industry is rapidly expanding, as drones prove to be valuable tools for automating tasks efficiently, [2]. Drones are used for various purposes, such as surveillance, delivery services, inspections, and maintenance. Applications for drones keep on increasing and will play a fundamental role in the near future performing a large number of tasks. An example can be seen in assessing climate change by facilitating ecosystem tracking, mapping, monitoring and data collection.

The seemingly expected growth comes at a cost, which is the concept once again of making these applications reliable, robust and safe. Ensuring that the airspace is free of conflicts is the key to the proper development of drone operations. In comparison to aircraft, drones are much smaller and their nominal speeds are low. This is beneficial as it solves the necessity of the large distances of separation that exist between planes for things such as wake vortices or conflict detection in all stages of flight but it does not strictly imply that safety is reduced. Similar measures can be taken for drones yet this still can result in the drone airspace being very densely populated with drones for relatively small volumes. This should pose no problem as long as everything works as intended and the design of the procedures are accordingly made to respect the safety guidelines. However, this does require the proper planning that ensures that there will be no conflicts over time and that safety will be preserved.

To address the challenge of ensuring safety, we rely on powerful tools capable of processing and predicting outcomes. Among the most notable and anticipated technologies, artificial intelligence stands out as an ideal solution for automation and prediction tasks.

With the power of artificial intelligence, we can effectively adapt to the dynamic changes that may arise during a flight, continuously reassess the situation, and maintain seamless operations without encountering any issues. Currently, drones are undergoing rigorous testing to observe their behavior, gather valuable data, and verify their ability to fulfill their intended purposes. This ongoing process of development and learning aligns perfectly with the scope of artificial intelligence, particularly in the field of Machine Learning. Machine Learning is dedicated to comprehending and constructing methodologies that enable machines to "learn" from data, ultimately enhancing their performance across a range of tasks.

In this study, our aim is to extract valuable features from an extensive dataset that encompasses flights conducted during the Very Large Demonstration (VLD) of CORUS-XUAM in Castelldefels in March 2022. CORUS-XUAM, a two-year VLD project, serves as a platform to showcase how U-space services and solutions can effectively support integrated Urban Air Mobility (UAM) flight operations. The project's main goal is to enable airspace users to operate in a controlled and fully integrated airspace, ensuring safety, security, sustainability, and efficiency throughout their operations.

This dataset comprises Flight Plans, Telemetry, and U-Space Predictions for each drone flight. To begin our analysis, we initially focus on comparing the intended trajectories with the actual flight paths, aiming to identify significant factors that contribute to deviations from the Flight Plan. This raw data necessitates processing to extract meaningful information, which can then be utilized in a Machine Learning model. The primary purpose of this model is to predict various parameters, including the telemetry time, based on a given Flight Plan.

U-Space emerges as a response to the increasing number of drones in the airspace, particularly in congested urban areas. With the goal of ensuring the safety and efficiency of drone operations, U-Space provides a range of key services and functionalities. These include drone traffic management, airspace access control, real-time monitoring, flight notification, and obstacle detection and avoidance, among others. The main objective of the U-Space concept is to establish a digitized infrastructure and airspace management that allows for the safe integration of drone operations into the existing airspace, without interfering with manned aviation operations, [2].

CHAPTER 1. DATA INSPECTION

1.1. Initial Data and structure

Data holds immense value in today's world, but its true potential can only be seen through appropriate utilization. Raw data, in its unprocessed state, lacks the capability to yield impressive outcomes. Therefore, it becomes essential to engage in a series of tasks such as data cleaning, processing, analysis, and modeling to extract the maximum advantage from it. This project centers around the comprehensive analysis and processing of drone data.

In conjunction with the project's main focus, we have available a collection of files that will serve as the foundational source for our data models, following the necessary processing steps. Specifically, these files originate from the Very Large Demonstration conducted in Castelldefels, which aimed to gather data from drones involved in delivery operations. Each trajectory took off from a designated delivery vertiport, reached another delivery port for the purpose of completing a delivery, and subsequently returned to the original port. Essentially, the drones' mission revolved around delivery operations to their assigned destinations.

However, it is crucial to recall that the trajectories followed by these drones have no resemblance to straight lines. Due to various factors, including the presence of airways operating at different altitudes than the delivery points, the drones must navigate through a network of flight paths. This aspect closely mirrors real-world scenarios of drone deliveries, particularly in urban environments where strategic airway designs are employed to avoid obstacles such as buildings, powerlines, or restricted areas. By adhering to these designated airways, drones can safely reach their intended destinations. In essence, these airways serve as the aerial equivalent of roads for ground traffic, but with the added vertical dimension to accommodate drone operations.

The dataset we possess captures a specific segment of the drone's overall path during its flights, rather than encompassing the entire duration from the moment it was turned on. This limitation arises due to the fact that the takeoff and landing procedures were manually executed by drone pilots, as opposed to being autonomously controlled by the drones themselves. Therefore, the recorded data begins at the precise instance when the drones reached the initial point of their intended flight plan trajectory.

The first and last points of these trajectories align directly above the delivery points, shifted at a certain vertical distance from them. It is worth noting that the delivery vertiports are situated at an elevated altitude above the ground level. To simulate the delivery operation, the drones hover at a specific altitude above the delivery points. During the hovering phase, the drone remains stationary in the exact position for an average duration of ten seconds. Although no actual delivery takes place in this context, the focus here lies in assessing the drone's ability to adhere to the predetermined flight plan.

1.1.1. Files

Among the provided files we can classify them in the following.

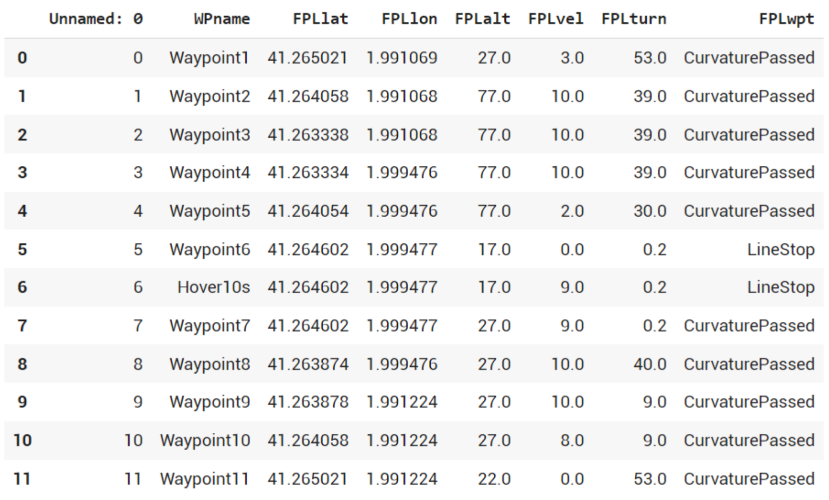
1.1.1.1. FlightPlan

The Flight Plan documents serve as the guidelines for the drone's designated route. Our dataset is made of processed .kml (Keyhole Markup Language) files, turned into .csv (Comma Separated Values) files, which provide a structured format for storing data. In this format, the Flight Plan information is organized into rows, with each row containing the properties of a specific point along the route.

To establish the trajectory, a series of waypoints are defined in three-dimensional space, represented by latitude, longitude, and altitude, this latter one above the vertiport. These waypoints act as reference points that define the desired path for the drone to follow. By connecting two consecutive waypoints, a Flight Plan Segment is formed, representing a distinct section of the overall route. Each segment is meant to be flown at a specific velocity, as indicated in the corresponding row for the origin point of that segment.

Additionally, the Flight Plan data includes a column specifying the Turn Radius (as the smoothing radius of the arc, in meters, taken by the drone in order to avoid an abrupt change in direction), which is the radius, measured in meters, that the drone should maintain when transitioning between segments. This parameter ensures that the drone maintains a smooth and controlled trajectory as it navigates from one segment to the next. Waypoints named as *Hover* are stationary stages of the flight which simulate the delivery. The drone must remain still for the duration specified in the waypoint name.

We are not going to make use of the columns *Unnamed: 0, WPname* (except for the information about the hover) and *FPLwpt*, as they do not provide information that we can benefit from.



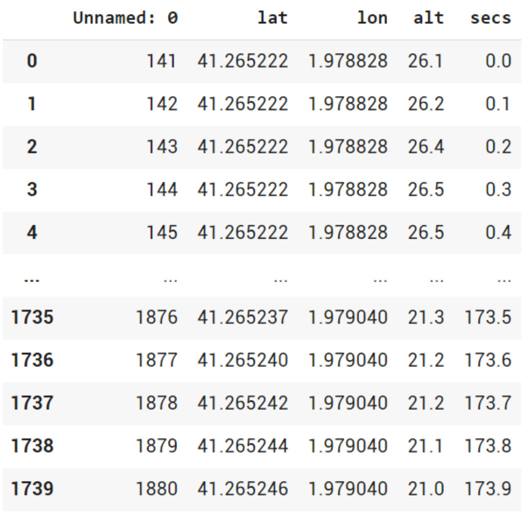
	Unnamed: 0	WPname	FPLlat	FPLlon	FPLalt	FPLvel	FPLturn	FPLwpt
0	0	Waypoint1	41.265021	1.991069	27.0	3.0	53.0	CurvaturePassed
1	1	Waypoint2	41.264058	1.991068	77.0	10.0	39.0	CurvaturePassed
2	2	Waypoint3	41.263338	1.991068	77.0	10.0	39.0	CurvaturePassed
3	3	Waypoint4	41.263334	1.999476	77.0	10.0	39.0	CurvaturePassed
4	4	Waypoint5	41.264054	1.999476	77.0	2.0	30.0	CurvaturePassed
5	5	Waypoint6	41.264602	1.999477	17.0	0.0	0.2	LineStop
6	6	Hover10s	41.264602	1.999477	17.0	9.0	0.2	LineStop
7	7	Waypoint7	41.264602	1.999477	27.0	9.0	0.2	CurvaturePassed
8	8	Waypoint8	41.263874	1.999476	27.0	10.0	40.0	CurvaturePassed
9	9	Waypoint9	41.263878	1.991224	27.0	10.0	9.0	CurvaturePassed
10	10	Waypoint10	41.264058	1.991224	27.0	8.0	9.0	CurvaturePassed
11	11	Waypoint11	41.265021	1.991224	22.0	0.0	53.0	CurvaturePassed

Figure 1. Generic raw Flightplan

1.1.1.2. Telemetry

In addition to the Flight Plan documents, we also have Telemetry files that capture the actual flown trajectory of the drones. These files provide valuable data on the drone's movement and position throughout its autonomous operation.

Telemetry data is collected starting from the moment the drone transitions into autonomous mode at the first point, or close, that is specified in the Flight Plan. It is a detailed record of the drone's flight, made up of latitude, longitude, and altitude coordinates. Notably, the Telemetry files are generated with a high frequency, recording the drone's position at a time step of 0.1 seconds. Consequently, these files tend to be significantly longer in comparison to the Flight Plan files.



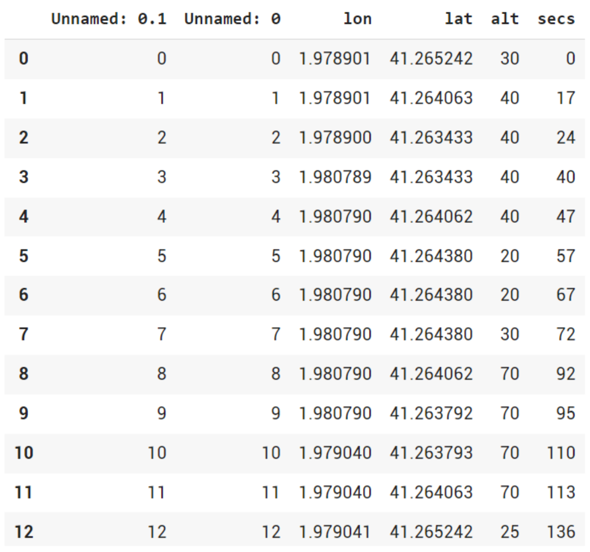
	Unnamed: 0	lat	lon	alt	secs
0	141	41.265222	1.978828	26.1	0.0
1	142	41.265222	1.978828	26.2	0.1
2	143	41.265222	1.978828	26.4	0.2
3	144	41.265222	1.978828	26.5	0.3
4	145	41.265222	1.978828	26.5	0.4
...	...	...	...	...	...
1735	1876	41.265237	1.979040	21.3	173.5
1736	1877	41.265240	1.979040	21.2	173.6
1737	1878	41.265242	1.979040	21.2	173.7
1738	1879	41.265244	1.979040	21.1	173.8
1739	1880	41.265246	1.979040	21.0	173.9

Figure 2. Generic raw Telemetry

Given the increased length of the Telemetry files, it becomes crucial to optimize how we handle and process this data. Efficient data processing techniques should be employed to avoid excessive computation times and ensure the timely extraction of valuable insights.

1.1.1.3. U-Space Predictions

In addition to the Flight Plan and Telemetry files, we have an additional set of files called U-Space Prediction files. These files closely resemble the Flight Plan files, containing the same information, however, they have an additional column that introduces a crucial element: the time prediction.



	Unnamed: 0.1	Unnamed: 0	lon	lat	alt	secs
0	0	0	1.978901	41.265242	30	0
1	1	1	1.978901	41.264063	40	17
2	2	2	1.978900	41.263433	40	24
3	3	3	1.980789	41.263433	40	40
4	4	4	1.980790	41.264062	40	47
5	5	5	1.980790	41.264380	20	57
6	6	6	1.980790	41.264380	20	67
7	7	7	1.980790	41.264380	30	72
8	8	8	1.980790	41.264062	70	92
9	9	9	1.980790	41.263792	70	95
10	10	10	1.979040	41.263793	70	110
11	11	11	1.979040	41.264063	70	113
12	12	12	1.979041	41.265242	25	136

Figure 3. Generic U-Space Prediction raw file

The time prediction column in the U-Space files indicates the estimated moment at which the drone will be located over each point specified in the Flight Plan. This temporal dimension will be compared with the machine learning time predictions and evaluate the reliability of both approaches, among other observations. Insights about the comparison of the U-Space time against the Telemetry will be done too.

While the original Flight Plan files do not include temporal data, they offer valuable information such as the waypoints and velocities. With these details, we can obtain an approximate estimation of the time it would take for the drone to reach each point along the trajectory.

1.1.1.4. Drone Models

We also have a file that links each flight to the corresponding drone model used. This information is crucial for our machine learning models as it allows us to leverage the unique features and associations specific to each drone model. By observing patterns within a particular model type, we can analyze their impact on flights flown with the same model while keeping other models unaffected. This insight enables us to build more accurate and less generalized models, optimizing our understanding of drone behavior and improving predictions.

1.1.2. Google Colab

Right from the very beginning this project requires the utilization of a programming language as a means to comprehend the information of the drone data files. Merely looking at the files would not give us a comprehensive understanding of the underlying information, as achieving any sort of results relies on the implementation of code. The process of data processing offers various approaches which depend on the nature of the data. For instance, if we were working with signal data, options like Matlab could be considered. Alternatively, popular data analysis languages like R and Java could also serve the purpose. However, Python is the chosen one for both for its prominence in the data sector and personal preference.

Python's user-friendly syntax and the vast and supportive community surrounding it make it an accessible language for programmers. Especially in the case of myself not being too familiar with it at the start of the project. Also, the collection of libraries available for Python significantly eases and facilitates numerous tasks. In this project, we rely on libraries such as Pandas, Numpy, Matplotlib, and Scikit-learn, which form the foundation of our data analysis and modeling.

Another aspect of our approach is the utilization of the interactive environment provided by Google Colab, built on top of the Jupyter Notebook infrastructure. Within Colab, we can effortlessly create, edit, and execute code cells, alongside the explanatory text and visualizations and allowing us to save the results of previous executions. Colab executes code in Google servers, providing cloud storage capabilities and pre-installed libraries and packages. The interactive nature of this environment grants us the luxury of visualizing, updating, and modifying code while it runs, greatly facilitating the data processing workflow. These advantages make Colab the most ideal environment for the construction of our project.

CHAPTER 2. DATA PROCESSING

2.1. General data overview

Once we import the drone data into Google Colab, our initial task is to get an understanding of the data and visualize its characteristics. In order to do this, visualizing the Flight Plans becomes an essential step. While we could also go for 2D plots that represent the evolution of the three coordinates along the route, the absence of a temporal dimension makes it harder to create Latitude, Longitude, and Altitude vs. Time plots. Therefore, our approach will involve plotting the Flight Plans in 3D, which offers a clear visualization of the drone trajectories and, importantly, provides insights into the structure and appearance of individual segments. Understanding the shape and distribution of these segments is crucial in comprehending how the drone navigates through the delivery areas and to see how the airways are common in all Flight Plans. In the following Figure 1, it is depicted a generic Flight Plan with the waypoints named in the order and the 2 long straight segments are the airways while the delivery (hover) happens midway in the flight.

The name of the flights that will appear in the figure are in the format of $day\_operator\_vertiport$. For instance, we can take an example flight:

$$A\_JUNO\_142\_DELV\_06$$

This represents a flight performed on day A, from the JUNO operator and its destination is the delivery vertiport 06. The number in between helps differentiate flights that might share day, operator and destination vertiport.

When we represent the flights three-dimensionally, it is important to note that the axis are scaled so it is easier to visualize the different segments. This is only for the sake of visualizing the data. In Figure 13, a non-scaled flight is represented after coordinate conversion.

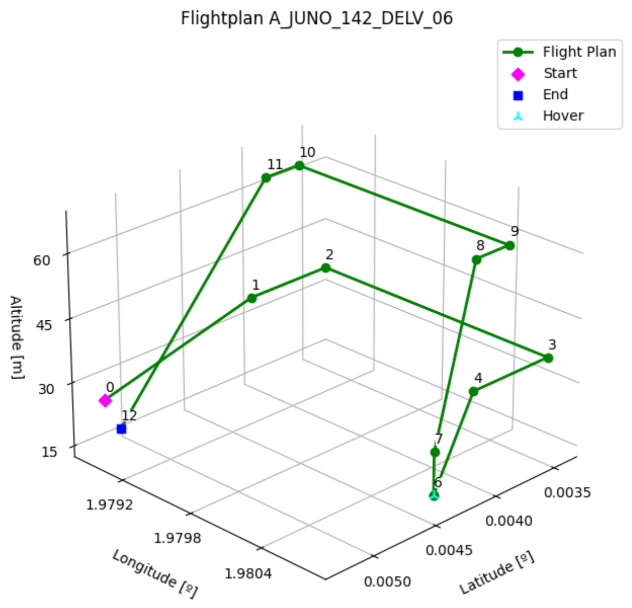


Figure 4. Representation of a FlightPlan file

Similarly, we can apply the same visualization approach to the Telemetry data of a flight. It is important to recall that the Telemetry data only provides information about the drone's location at a specific point in time. As depicted in Figure 5, we observe that there are no clearly defined segments in the Telemetry plot, and the overall trajectory appears to be smooth with seamless transitions between segments. This Telemetry representation corresponds to the Flight Plan showcased in Figure 4 and thus the remarkable similarity to it. Nevertheless, it is crucial to emphasize that the Telemetry data lacks a temporal component, therefore, with simply the 3D representation it is challenging to determine whether the drone adhered to the Flight Plan accurately in terms of time.

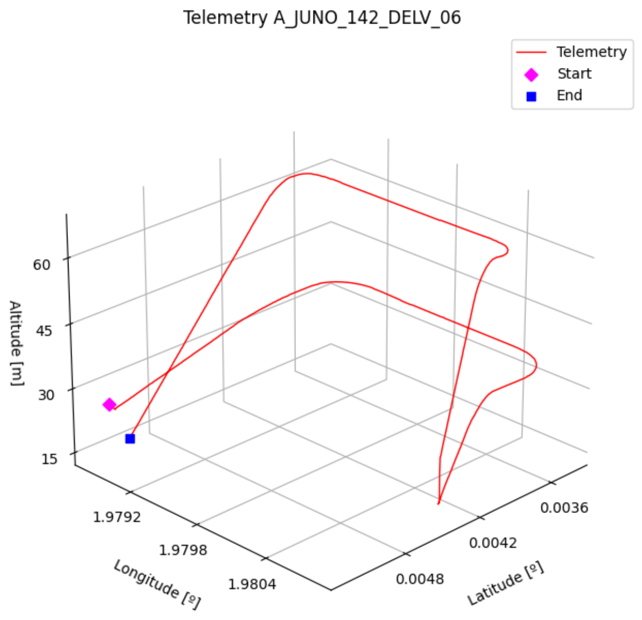


Figure 5. Representation of a telemetry file

While the spatial representation provides valuable insights into the drone's trajectory and spatial coherence with the Flight Plan, evaluating temporal alignment becomes a more complex task. Without explicit temporal information in the plots, it becomes difficult to determine if the drone followed the Flight Plan according to the temporal dimension. Even though the Flight Plan does not explicitly provide the time, it indicates constant velocity for the segments, which essentially dictates the time. Because of that, further analysis and techniques are necessary to assess the temporal synchronization between the Telemetry data and the corresponding Flight Plan. Regardless of that, we can still benefit from the 3D representations for exploring the data and finding flights that did not follow the Flight Plan.

An important aspect of our analysis involves plotting both the Flight Plan and Telemetry data in the same visualization, allowing us to assess how closely the drone adhered to its intended trajectory. This comparison between the Flight Plan and corresponding Telemetry will be a typical study throughout the project, as we continuously aim to identify similarities and differences between the two datasets. By juxtaposing these plots, we can easily locate instances where drones may have encountered issues or deviated from the prescribed trajectory. This straightforward approach provides us with valuable insights, enabling us to efficiently identify any anomalies that might be left out from the models to be used lately in this project.

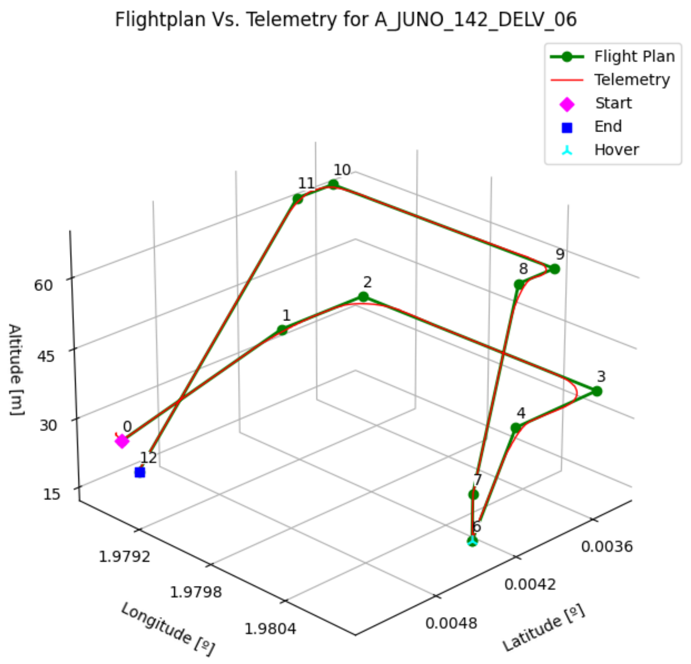


Figure 6. Representation of a FlightPlan and a telemetry file in the same plot.

In Figure 6, we can clearly observe a strong correlation between the telemetry trajectory and the intended trajectory, indicating that the drone performed with an excellent spatial performance during the flight. The smooth transitions between segments are guided by the specified Turn Radius according to the Flight Plan.

However, it is important to note that not all flights showcased such a close alignment between the Telemetry and Flight Plan. In multiple cases, we observe significant deviations and incomplete trajectories, suggesting that some drones may have experienced distortions or failed to follow the intended path accurately. These deviations in the telemetry data highlight the need for further investigation and analysis to identify these affected flights and later in the project we will assess a way to detect these cases without having to decide based on the plots by human eye. In Figure 7 we can visualize 4 flights whose telemetry did not perform accordingly to the planned one.

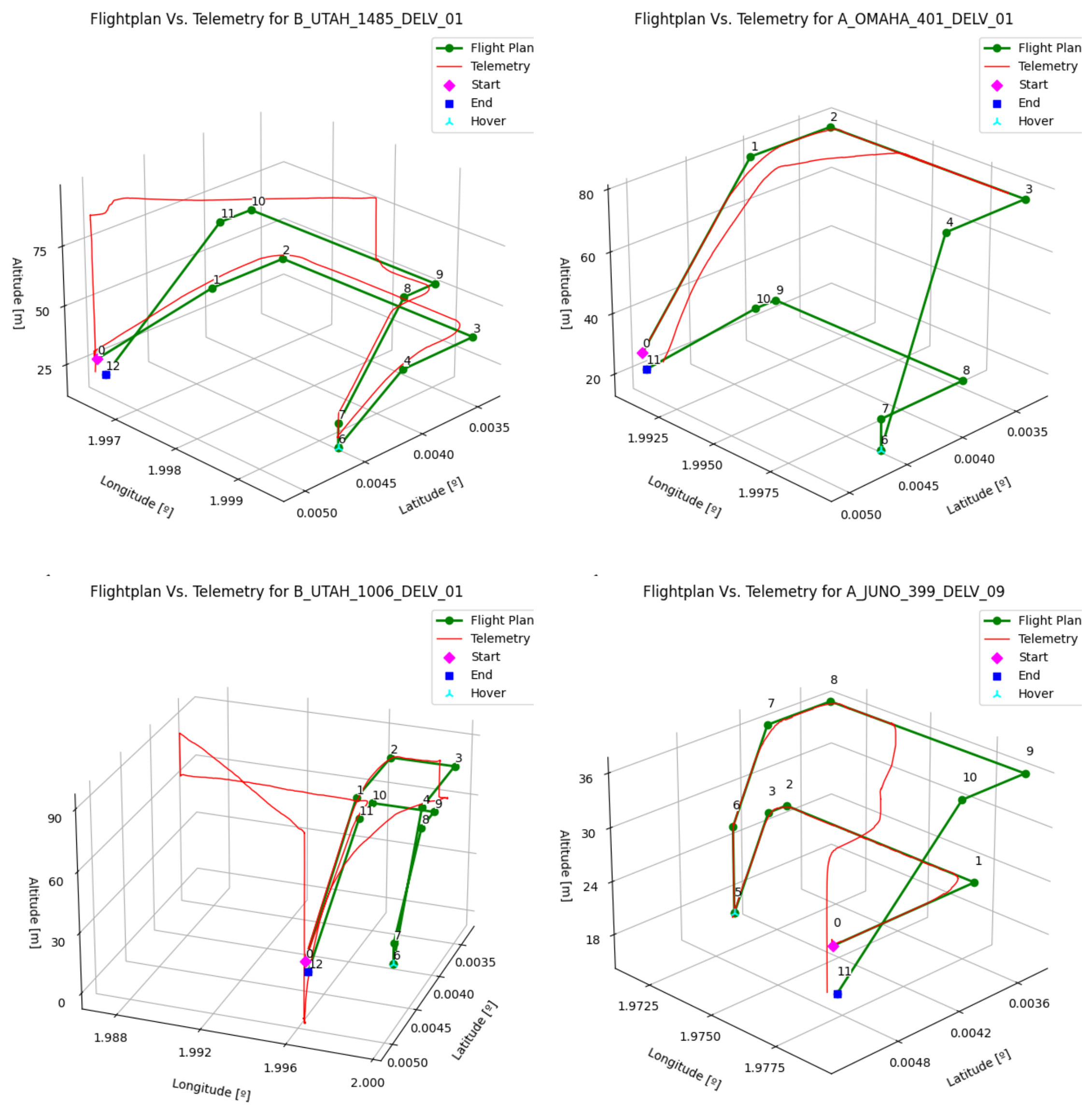


Figure 7. Example of contingency flights. See Annex for the plots of all contingency flights.

The deviations and anomalies observed in the flight trajectories cannot be definitively attributed to a specific cause, as the Telemetry data only provides spatial and time information.

However, several factors could potentially contribute to these behaviors. For instance, the properties of the drone itself may cause the deviation from the intended route. Factors such as low battery levels, overheating, loss of signal, or irregular performance of mechanical components could make the drone abort the planned trajectory. It is also possible that the pilot intervened and took remote control of the drone for various reasons. Additionally, external factors such as wind conditions or the presence of other air traffic may have played a role in some of these situations. Given the variety of data at our disposal, further comprehensive analysis can be performed to learn more about the nature of the flights and evaluate them.

It is important for us to gain an understanding of the data by examining the general context and factors that can influence the flights. Each drone operator has the flexibility to operate their fleet according to their own preferences, while always respecting the guidelines and regulations. This approach among operators increases the variability in deviations or distinct behavioral patterns in certain flights. But it is not just the operator as in the context of the Very Large Demonstration, not all flights from the same operator were conducted using the same drone model. Therefore, when analyzing the data and identifying deviations, it is essential to consider both the drone operator and the specific drone model employed, as these factors can contribute to the observations.

As we can observe in Figure 8, the number of flights performed by each operator is not too homogenous and OMAHA is leading with 31 performed flights in the VLM, followed by 20 flights from UTAH, 17 from JUNO and just 4 from SWORD. This gives us a total number of 72 flights, each with their respective Flight Plan and Telemetry.

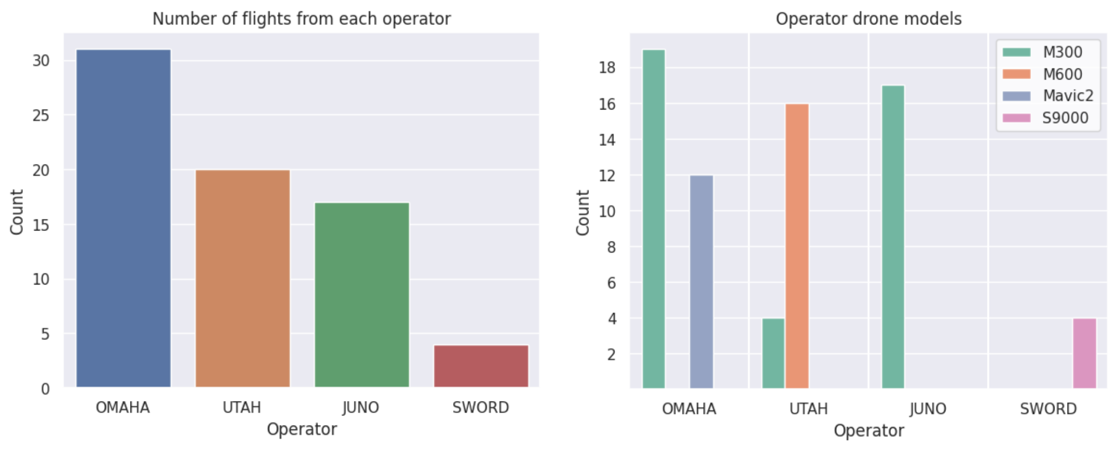


Figure 8. Count of flights by drone operators and drone model used

An important consideration in our data analysis is the quantity of data available. In our case, we have a database consisting of 72 sets of files, which raises the question of whether this amount of data is sufficient to achieve our objectives.

2.2. Basic data processing

Now that we have a general data overview, we can start to manipulate it to further see the nature of it and get interesting features from the data in the files. There are multiple parameters to compute from the telemetry as well as from the Flight Plan. The first test will be to get an idea of the U-Space time column, supposedly predicting the time stamps of the telemetry when it should overfly each waypoint. We can represent these as 3 subplots, Latitude, Longitude and Altitude versus Time. Representing these and comparing them to the Telemetry will be very helpful to visualize how accurate the U-Space predictions are.

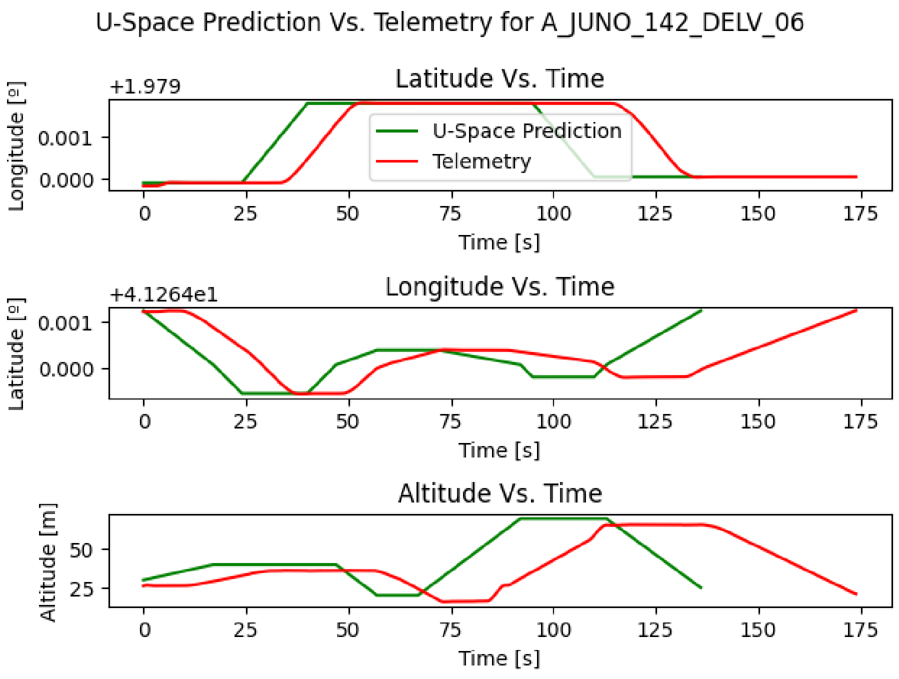


Figure 9. Coordinates over time for the U-Space Prediction and Telemetry

After examining the visualization in Figure 9, we notice a distinct temporal shift between the U-Space Prediction coordinates (plotted in green) and the Telemetry coordinates (plotted in red). This becomes apparent as the telemetry data appears to experience a delay of a few seconds before initiating the flight trajectory. It gives the impression that the drone is simply hovering still before initiating its movement.

To address this misalignment between the U-Space Prediction and Telemetry, one possible solution could be to exclude the portion of the telemetry data when the drone is stationary. Removing this period of inactivity from the dataset would improve the alignment between the U-Space Prediction and the actual drone trajectory, reducing the temporal shift observed. This is important considering that when analyzing both plots, we need to know whether we consider time or not. Without the time component, the trajectories are much more coherent as seen in the 3D plots, while when plotting the three separate subplots it looks completely different. Recall that the represented flight in Figure 1 in 3D is the same as the one in the subplots from Figure 9.

This puts into perspective the effect that time has on the trajectories and despite seeming accurate in 3D, the trajectories might not be even close to matching in 4D.

While it seems obvious to just adjust the telemetry delay by subtracting or adding (or cropping the file) a constant value to all the telemetry and shifting it to match the first point of the Flight Plan to the first point when the drone initiates the movement, it might not be assumed to be as straightforward as that in other cases.

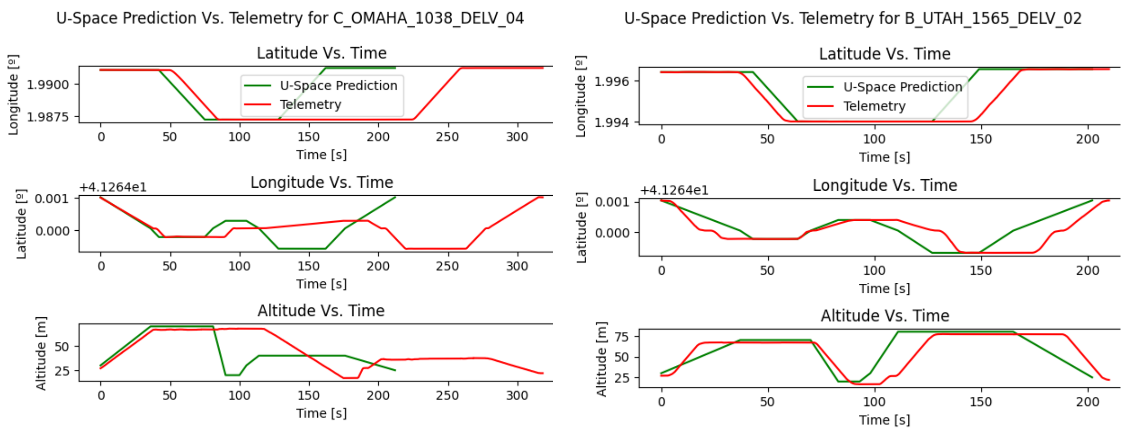


Figure 10. Coordinates over time for two cases where the shape of the curves is not easily fixable by correcting a delay.

Figure 10 presents an example where the discrepancy between the telemetry and the expected flight trajectory is more complex than a simple temporal delay. Multiple factors, as discussed previously, might have affected how the drone flew its route and caused these distinctive anomalies in the telemetry data. On the left side of the figure, the telemetry initiates at the expected time, aligning with the start of the intended route. However, as the flight progresses, an irregularity appears around the midpoint of the route as the telemetry from the drone tells that it might have slowed down, stretching the shape of the Flight Plan before resuming its normal pace.

In instances like these, it becomes apparent that introducing a time delay adjustment may not provide a straightforward solution for achieving improved accuracy. The issue here falls in the irregular patterns experienced by the drone which are then presented in the telemetry data, which cannot be easily rectified by simple temporal adjustments. If some of these irregularities happen unusually, we might consider taking them out of the model's training database as these kinds of behaviors are out of the usual performance of the drone and adding corrupted or excessively bad-performing flights can negatively affect the accuracy of the predictions later on.

As we don't know how the U-Space Prediction gets the time of the flights that it processes, we will do some testing that hopefully gives us some idea about the temporal aspect of the flight plans.

In order to assess the adherence of the telemetry data to the flight plan, we need some criteria that brings us more information than spatial correlation alone. Analyzing the velocity of the telemetry data based on its time steps could provide useful insights, however, the variability of drone velocity in real-life trajectories makes this approach less reliable. As seen in Figure 10, anomalies in telemetry data will often coincide with variations in velocity, making an analysis based on velocity might not be the best due to noise and not the most intuitive comparison method.

To get around this, we can take advantage of the simplicity and structure of the flight plans to obtain a more robust temporal reference. With the spatial coordinates and the corresponding velocity between two points in the flight plan, we can mathematically estimate the time stamps for each point in the flight plan dataframe. This approach allows us to create a time reference for the flight plan that can serve as a basis for comparison with the telemetry data. This method is far more comprehensive as we assign the theoretical time to each waypoint and from that, we can have references for the telemetry.

From now on, the data files that we have at our disposal will be referred to as *dataframes*, a term that defines a data structure resembling a matrix, organized in rows and columns. With dataframes, we have the ability to perform various operations, such as targeting specific cells, rows, or columns, as well as applying conditions to locate and extract desired data. The information stored within these cells is not limited to numeric values. It includes a wide range of data types, including strings, series, arrays, lists, objects, and more. To do this, we will be using Pandas library, specifically designed for these purposes and integrated with Python. Pandas contains the pandas.DataFrame class, as the primary form of structuring and organizing data [3]. We will be using this library for the whole project and it will facilitate us all the tasks.

To compute the time from a Flight Plan, we need to first get the distance between points. We consider that the way the drone travels between two points is in a straight line. However, to get the distance, we make a conversion of the coordinate system, from geodetic coordinates (latitude, longitude and altitude) to the Earth-centered, Earth-fixed coordinate system (ECEF). This will make it easier to work with the position of the waypoints as we have the same units of magnitude in each component of the coordinate and we can then use math operations without worrying about the conversion.

Recall that the altitude that the Flight Plan gives us, is different from the Z coordinate of the ECEF system. The altitude provided by the plan is with respect to the take-off point.

The ECEF coordinate system represents points in the form of X, Y, Z with respect to the center of mass from the Earth:

- Positive X axis lies on the equator, from the mass center towards the direction of the Greenwich Meridian.
- Positive Y axis also lies on the equator, 90º to the East of the X axis (Greenwich Meridian)
- Positive Z axis extends northwards, to the North Pole, from the center of mass.

$$\boldsymbol{X} = (\boldsymbol{N}(\boldsymbol{\phi}) + \boldsymbol{h})\boldsymbol{cos\phi cos\lambda} \quad \textbf{(Eq. 1)}$$

$$\boldsymbol{Y} = (\boldsymbol{N}(\boldsymbol{\phi}) + \boldsymbol{h})\boldsymbol{cos\phi sin\lambda} \quad \textbf{(Eq. 2)}$$

$$\boldsymbol{Z} = (\frac{\boldsymbol{b^2}}{\boldsymbol{a^2}}\boldsymbol{N}(\boldsymbol{\phi}) + \boldsymbol{h})\boldsymbol{sin\phi} \quad \textbf{(Eq. 3)}$$

$$\boldsymbol{N}(\boldsymbol{\phi}) = \frac{\boldsymbol{a^2}}{\sqrt{\boldsymbol{a^2cos^2\phi}+\boldsymbol{b^2sin^2\phi}}} \quad \textbf{(Eq. 4)}$$

Where ϕ is the latitude, λ the latitude, h the altitude and $N(\phi)$ represents the prime vertical radius of curvature [4].

As for the parameters that define the shape of the ellipsoid, we take the World Geodetic System 1984 (WGS 84) model as it is standardized and widely used for services such as Global Positioning System (GPS) [5]:

- Semi Major Axis, a = 6378137 m
- Semi Minor Axis, b = 6356752.31424 m

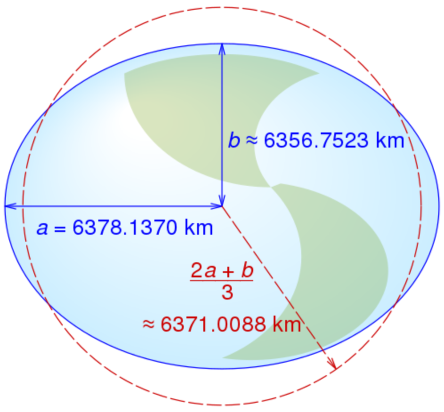


Figure 11. ECEF, [6]

After this conversion it does not get that much more intuitive as we are dealing with huge distances and the movement of the drone in comparison to the baseline distance of a point located above the surface of the Earth is incredibly small.

However, now the three coordinates share the same units and operations become easier and possible. In addition, we might have negative distances. For instance, if the drone moves South, from Castelldefels, that reduces the Z coordinate. If we then do operations such as subtracting the end point to the origin point of a segment, it can yield negative results. Distances can be considered in magnitude as always positive but the sign in this case will indicate the direction and it is helpful to keep it in. In Figure 11, we can see a Flight Plan plotted in X, Y, Z coordinates.

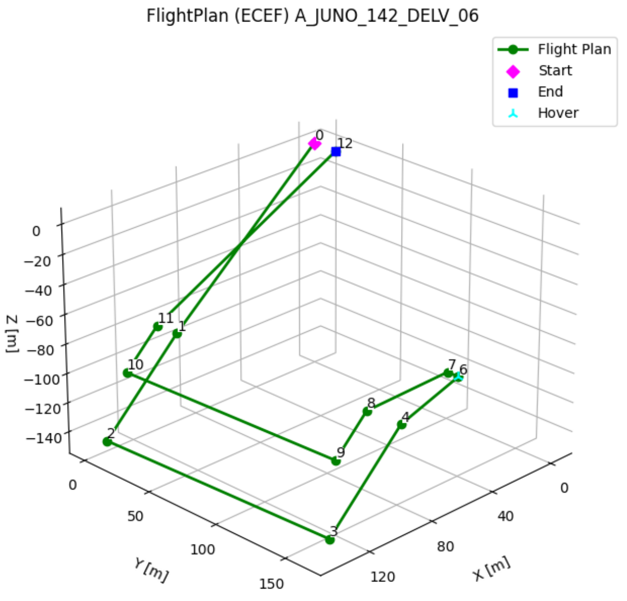


Figure 12. Flight Plan representation after conversion from geodetic to ECEF coordinates

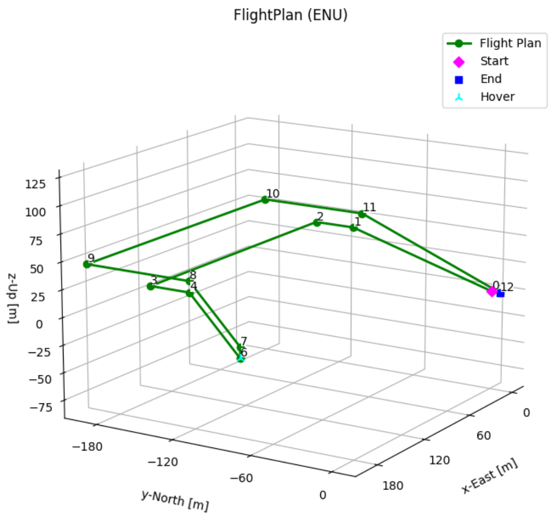
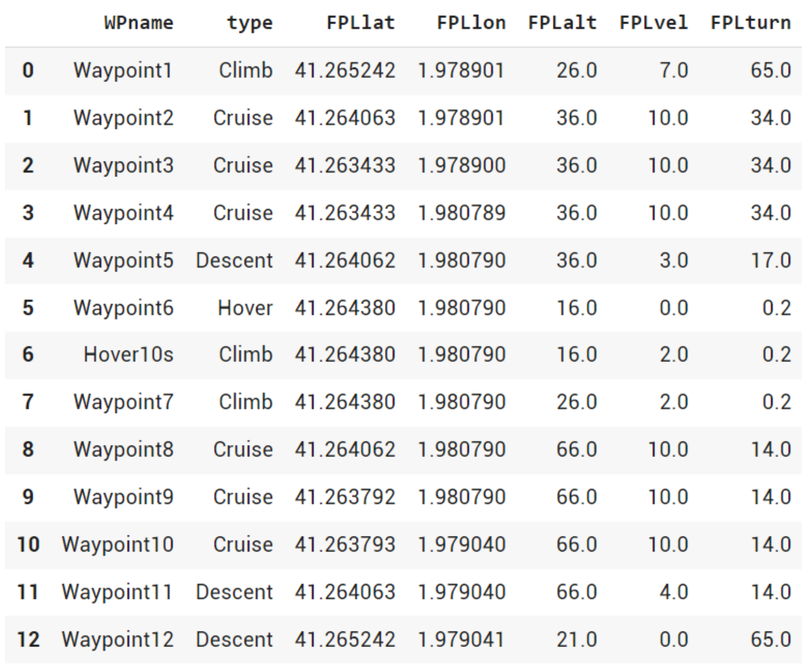


Figure 13. Representation of a FlightPlan in ENU coordinates, [6]. Important to note that this figure show the real scale of a flight, unlike others that have their axis scaled for easier visualization.

Having done this conversion, it may appear hard to tell what kind of segment the drone is actually performing by just looking at Figure 12. It will appear that during the first segment, from waypoint 0 to waypoint 1, the drone is descending but it is actually the opposite. To categorize the segments based on the data that we have, it is simpler to do it in the geodetic coordinates before conversion. Determining the segment type is going to be useful for the model later. This can be done by simply comparing altitudes. If the altitude of the end point of a segment is lesser than than the altitude of the origin point, we are dealing with a descent. On the other hand, if it's greater, it will be a climb. If the altitude remains the same, it can be defined as a cruise segment. And finally for the hover stages, we take a look at the information of the waypoint in the Flight Plan and see that it is defined by a hover and its respective time.



	WPname	type	FPLlat	FPLlon	FPLalt	FPLvel	FPLturn
0	Waypoint1	Climb	41.265242	1.978901	26.0	7.0	65.0
1	Waypoint2	Cruise	41.264063	1.978901	36.0	10.0	34.0
2	Waypoint3	Cruise	41.263433	1.978900	36.0	10.0	34.0
3	Waypoint4	Cruise	41.263433	1.980789	36.0	10.0	34.0
4	Waypoint5	Descent	41.264062	1.980790	36.0	3.0	17.0
5	Waypoint6	Hover	41.264380	1.980790	16.0	0.0	0.2
6	Hover10s	Climb	41.264380	1.980790	16.0	2.0	0.2
7	Waypoint7	Climb	41.264380	1.980790	26.0	2.0	0.2
8	Waypoint8	Cruise	41.264062	1.980790	66.0	10.0	14.0
9	Waypoint9	Cruise	41.263792	1.980790	66.0	10.0	14.0
10	Waypoint10	Cruise	41.263793	1.979040	66.0	10.0	14.0
11	Waypoint11	Descent	41.264063	1.979040	66.0	4.0	14.0
12	Waypoint12	Descent	41.265242	1.979041	21.0	0.0	65.0

Figure 14. Dataframe with segment type added into dataframes as column 'type'

Similarly, we will take the same approach for the model of drone utilized for every flight. The values in every row will be the same for a given Flight Plan as the model of drone doesn't change along the trajectory. For the model to learn though, the values do not work with simply keeping them as strings. We should rather convert them to integers for each kind of segment and drone or do one-hot-encoding.

As mentioned, now that we have the ECEF coordinates for the points, it is possible for us to start computing distances between segments. To compute the distance between two points in 3D space, it is relatively easy with the following formula.

$$\boldsymbol{3D\ distance} = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2 + (z_1 - z_0)^2} \qquad \textbf{(Eq. 5)}$$

To determine the number of segments in a Flight Plan, we can apply a straightforward formula. For every Flight Plan, the number of segments is equal to the number of points minus one. This relationship comes from the fact that each segment is formed by a pair of consecutive points rather than individual points. Two points form a segment, not two. By connecting these paired points, we establish the trajectory of the drone throughout its flight.

Mathematically, we can express this relationship as:

$$n_{segments} = n_{waypoints} - 1 \quad \textbf{(Eq. 6)}$$

For the dataframes containing the segments, it is true that they will have a row less than the Flight Plan, however, the number of points will remain the same. Each row consisting of a segment will own two points: an origin point and an end point that will serve as the origin point for the next segment.

```
## convert and replace the lat, lon, alt to x, y, z
for i, row in flightPlan.iterrows():
  flightPlan.at[i,'FPLlat'],flightPlan.at[i,'FPLlon'],flightPlan.at[i,'FPLalt']= \
  geodetic2ecef(row['FPLlat'],row['FPLlon'],row['FPLalt'])

## compute the segment time by dividing the absolute distance by the velocity
for (_, original), (i, shift) in zip(flightPlan.iloc[:-1].iterrows(),
flightPlan.iloc[1:].iterrows()):

  if  (original['FPLvel']) > 0:
    dist=math.sqrt((shift['FPLlon']-original['FPLlon'])**2+
     (shift['FPLlat']-original['FPLlat'])**2+(shift['FPLalt']-original['FPLalt'])**2)
    flightPlan.at[i, 'timeSeg']=dist/(original['FPLvel'])

  else:
    flightPlan.at[i, 'timeSeg'] = int(flightPlan.at[i,'WPname'].replace(
        'Hover', '').replace('s',''))

## fill NaN of the first row (as it is not considered in the loop) and get global time
flightPlan['timeSeg'].fillna(0, inplace=True)
flightPlan['time']=(flightPlan['timeSeg'].cumsum(axis = 0))
```

Listing 1: Computation of time associated to the FlightPlan

In the Listing 1, we can see a part of the function of Colab that computes the time when the drone should be reaching every waypoint. The procedure is intuitive as we start by iterating row by row with the Dataframe.iterrows() method provided once again by Pandas library. With this method the iterations provide an index and the row information attached to that index. This way we can use both the index and row content comfortably to our needs.

In this case, it is used to apply the function of coordinate conversion *geodetic2ecef* to every single row as it works with 3 inputs and provides 3 values.

In the development of this project, we also make extensive use of the zip() function in Python to create a powerful and highly useful data structure. The zip function allows us to combine iterables, such as dataframes, and pair them together in tuples. This functionality is extremely valuable when operating on multiple rows simultaneously while keeping track of their respective indices, allowing precise data allocation and manipulation.

Specifically, when computing the time between segments in the Flight Plan, we take advantage of said zip function. We select two rows from the dataframe: one representing the origin point and the other representing the end point of the segment. By subtracting their corresponding coordinate and applying Equation 2 we obtain the distance between these points. This distance is then divided by the velocity provided in the row containing the origin point. This calculation yields the time required for the drone to fly the segment. When it comes to hovering segments within the route, we directly extract the time from the waypoint definition specified in the Flight Plan. By summing up the individual segment times cumulatively, we obtain the total time it should take for the drone to complete the entire flight route.

Now we can observe how our computation of the time resembles the U-Space time prediction and see if we can deduce any useful information.

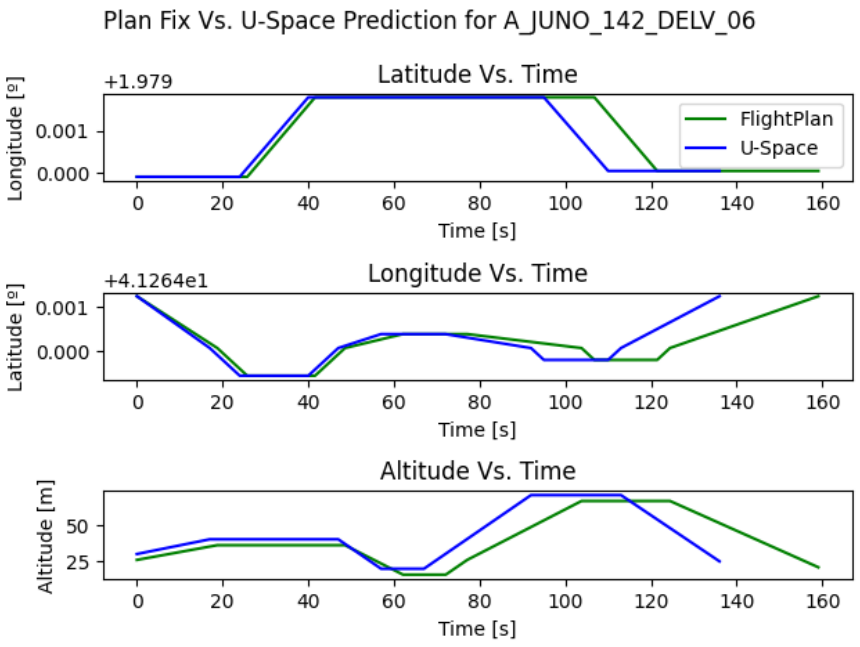


Figure 15. Comparison of the mathematical approach for a Flight Plan time versus the U-Space time prediction

Observing Figure 15, it is clear that the U-Space predictions do not represent the exact mathematical time estimation for each point, or at least it is not like this for all files.

The coordinates in the U-Space predictions seem to remain unchanged, with only a temporal stretch applied to them. To assess the accuracy of these different time calculations, we can compare them against the telemetry data of the flight. This allows us to determine which time estimation method yields better results, or at least for a subset of flights to get a first glance. However, it is important to note that we can’t generalize these findings, as each flight can has its unique characteristics and variations.

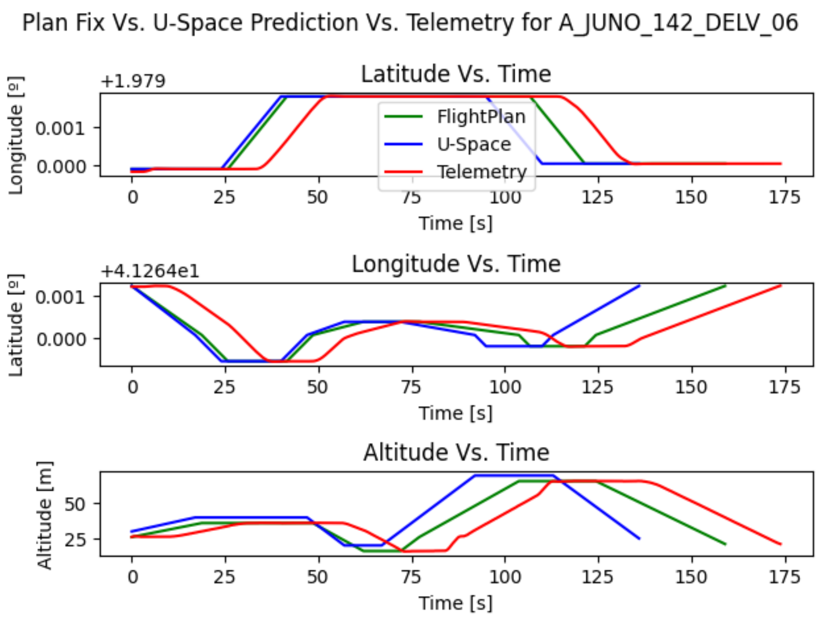


Figure 16. Comparison of the mathematical approach for a Flight Plan, the U-Space Prediction and the Telemetry

For instance, in Figure 16, we visualize the data from the three provided files for a given flight. In this case, there is evident variation between all of them and none of the update Flight Plan or U-Space prediction seems to closely match the telemetry. Yet if we put special emphasis on time prediction, the mathematical approach appears to have performed better, as the error on the final time of the U-Space seems to be more than twice (around 40 seconds) as the error from the mathematical approach on the Flight Plan, that is around 15 seconds. We could however also assess spatial accuracy, and in this sense it is not as clear on which of both trajectories resembles more the telemetry. A way of numerically computing these values can be seen in [7].

By observing other representations of other flights, we get to the conclusion that there is indeed a high level of variation and unpredictability to these results. As seen in Figure 17, there are instances where both the mathematical approach virtually matches the U-Space predictions, seen in the left upper corner. In other cases the mathematical approach is way off from the telemetry, while in other cases it is the U-Space.

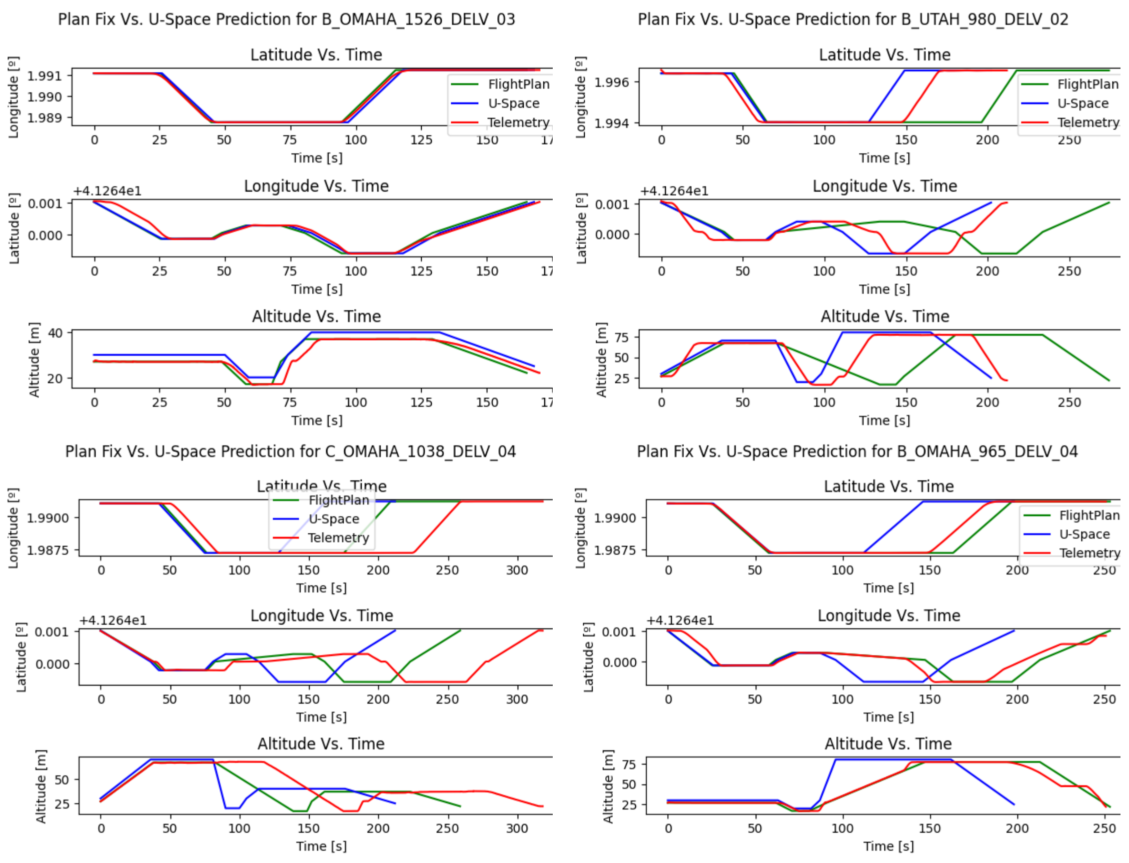


Figure 17. Different cases comparing the three files. High variation.

It is true that in an ideal scenario, where the drone maintains a constant speed and follows a straight line with the designated turn radius to each waypoint, the telemetry data should closely align with the time computed in Listing 1. This computed time takes into account the regular behavior of the drone based on the Flight Plan waypoints. Even though the telemetry of the drone flights deviates from this theoretical one due to various factors, it is still valuable to consider the computed time for future comparisons and analysis. The computed time provides a consistent and predictable reference point, allowing us to assess the temporal accuracy and deviations in the telemetry data. This temporal reference and the suitable coordinate system, we establish a foundation for further analysis and exploration in the project.

2.3. Time independent error analysis

Assessing the accuracy and performance of a drone along its intended route partly involves analyzing its adherence to the Flight Plan. With the deviations from the intended path, independent of time, we can evaluate how precisely the drone follows the intended trajectory. These deviations introduce additional distance traveled, as the drone moves away from the straight line connecting the points of each segment. The more significant the variations and noise in the trajectory, the greater the distance the drone has to cover, potentially resulting in longer flight times.
Understanding the spatial aspects of the drone's trajectory is a key factor for evaluating its adherence to the Flight Plan. If the drone was perfectly designed to follow the exact trajectory at a constant speed, we would have a matching telemetry and flight plan. This analysis allows us to identify areas where the drone may deviate from the intended path, indicating potential issues or anomalies in its flight behavior or weak points in the drone’s auto-pilot system in particular shapes of the trajectory.

In order to evaluate the deviations between Telemetry and Flight Plan, a careful manipulation of the dataframes is required. The approach taken in this analysis is primarily focussed on the telemetry points and establishing their association with the corresponding segments of the Flight Plan. However, this association of telemetry points to specific segments is not trivial during transitions between segments. The assignment based on the time is not reasonable, as we have observed in previous observations that there were significant variations between telemetry and Flight Plan time estimations, resulting in considerable deviations when the drone is either ahead of or behind the expected trajectory.

Therefore, an alternative approach is used to address this issue. Instead of relying on time-based matching, other factors such as spatial correlation and proximity are considered for the associations between telemetry points and Flight Plan points. As we want to develop a robust function to compute this, that could work in any sort of trajectory despite knowing that the ones at our disposal are relatively simple, we will make use of a method that does not actually consider the segments as parts of the route where a given section of the telemetry has to be associated but rather considering all the trajectory as a whole and comparing it to the whole intended trajectory of the Flight Plan. This holds a similitude to signal correlation, where the analysis of correlation between signals is important for synchronization and to reduce the effect of the noise. For this sort of analysis, having a high amount of samples is essential and we will in our case increase the number of samples (points) in our Flight Plan so as to have more anchor reference points to attach the telemetry to.

```
## Flight Plan extension
df_flightPlanExpanded = pd.concat(
    [
        pd.DataFrame(
            scipy.interpolate.interp1d([0, 1],[[original["FPLlat"], original["FPLlon"],
original["FPLalt"]],[shift["FPLlat"], shift["FPLlon"], shift["FPLalt"]]], axis=0,
Bounds_error=False,fill_value=[0, 0, 0])(np.linspace(
0, 1, 50)), columns=["FPLlat", "FPLlon", "FPLalt"])
        for (_, original), (_, shift) in zip(df_flightPlan.iloc[:-1].iterrows(),
df_flightPlan.iloc[1:].iterrows())
    ]
)

df_flightPlanExpanded = df_flightPlanExpanded.reset_index(drop=True)

## Lat, Lon, Alt to x, y, z with geodetic2ecef
for index, row in df_flightPlanExpanded.iterrows():
  df_flightPlanExpanded.at[index,'FPLlat'],df_flightPlanExpanded.at[index,'FPLlon'],_=
geodetic2ecef(row['FPLlat'],row['FPLlon'],row['FPLalt'])

for index, row in df_telemetry.iterrows():
  df_telemetry.at[index,'lat'],df_telemetry.at[index,'lon'],_=
geodetic2ecef(row['lat'],row['lon'],row['alt'])
```

Listing 2. FlightPlan interpolation and coordinate conversion

First we are going to increase the number of points from the FlightPlan trajectory. The trajectory will remain the same but with a higher number of points that will be used to compute their distance to the telemetry path. The idea is to take two consecutive points of the FlightPlan and get the function of the line that connects them.

This way, we have a function f(x), where:
- $f(0) = origin\_point$
- $f(1) = end\_point$
- $for\ 0 \leq x \leq 1$

We don not take into consideration values outside the range of x ∈ [0,1] as we only want to add points in the imaginary line that links two consecutive points of the FlightPlan, this is adding points only in-between the origin point (for x=0) and the end point (for x=1) of every segment that makes the FlightPlan.

Essentially, with an origin_point = (lat0, lon0, alt0) and an end_point = (lat1, lon1, alt1).

- $FPLlat(x), where\ FPLlat(0) = lat0\ and\ FPLlat(1) = lat1$
- $FPLlon(x), where\ FPLlon(0) = lon0\ and\ FPLlon(1) = lon1$
- $FPLalt(x), where\ FPLalt(0) = alt0\ and\ FPLalt(1) = alt1$

Therefore, we can take as many values x as we want to get points in the line as long as 0 ≤ x ≤ 1, so for instance, considering that the FlightPlan is made by straight lines connecting the waypoints, FPLlat(x) = (lat1 - lat0) * x + lat0 so if lat0=0 and lat1=10, FPLlat(0.5)=5, as expected, the middle point.

In order to do this, what we are doing is to use *scipy.interpolate.interp1d* from the scipy library. As the name suggests, this is a function that finds a function(s) from given values of x and y. We are just providing two points so the interpolation is perfect, as by default this function does linear interpolation and by providing two points, it simply creates the line that contains the two points.

Our first objective is to increase the number of points in the FlightPlan, in this case, we will add 50 points in each segment, so for a FlightPlan determined by 12 waypoints, we will end up with 12*50=600 points. This is done by iterating the interpolation function *scipy.interpolate.interp1d* 50 times with values from 0 to 1, using *np.linspace(0, 1, 50)* we get an array from 0 to 1, both included, with 48 other extra equally-spaced values in between.

We have to provide the function of the two points, origin and end, which are consecutive points (rows) in df_flightPlan. In order to do this, we simply create a shifted copy of df_flightPlan that starts a point later than the original df_flightPlan with the use of *df_shift = df_flightPlan.iloc[1:].* Doing this, reduces the length (rows) of the dataframe by one, so *len(df_shift)=len(df_flightPlan)-1*. Therefore, we need to trim the unshifted (df_original) dataframe by one row (the last one) so that both of them have the same length using *df_original = df_flightPlan.iloc[:-1].*

Now, for the same index (same row number) in both dataframes, we will get the origin point and end point so that origin will be in *original.iloc[0]* and the end in *shift.iloc[0].* This comes in handy to use a for loop as we are always using the same row index for both dataframes. To get the 50 points, we need to just provide a single origin_point and also a single end_point to the interpolate function. Providing the whole dataframe would create a line that is close to all waypoints, like a regression line and that's not what we want. In order to do this, we have to use *scipy.interpolate.interp1d* as many times as segments exist in the dataframe.

The number of segments is just obtained by subtracting 1 to the number of waypoints in the FlightPlan, which is essentially what has been done before in the df_original and df_shift dataframes. Then, in each iteration, we take a row of each dataframe, get the origin_point from the df_original and the end_point from df_shift, with these two we obtain the line equation(s) *f(x)* to which we will pass the 50 numbers from 0 to 1, this is *f(np.linspace(0, 1, 50))* where *f=scipy.interpolate.interp1d([0, 1],[[origin_point],[end_point])* with the rest of parameters needed for the function (axis, fill_value...).

This will give us 50 points in-between the first two consecutive points of df_flightPlan. We are creating a new dataframe *df_flightPlanExpanded* which will contain all these groups of 50 points generated in each iteration, so in every iteration we are appending 50 points to it. This can be done with *pd.concat*.

All the iterations are done with *pandas.DataFrame.iterrows* which, as the name suggests, iterates over the rows. We are using *zip* which pairs, in tuples, the items of the passed iterators. So the first item in each passed iterator is paired together, and then the second item in each passed iterator are paired together, etc. Put simply, in each iteration, it will create a tuple storing 2 other tuples, each of these tuples contains two items: an integer (int) being the index of the row and a series (pandas.Series) containing the information about the row (lat, lon, alt...).

$zip \rightarrow (tuple1, tuple2) \rightarrow ((int1, series1), (int2, series2)) \rightarrow$
$((int1, [WPname1, F \quad PLlat1, FPLlon1...]), (int2, [WPname2, FPLlat2, FPLlon2...]))$

We don't need the first items from each tuple (int1 and int2) as it is just an integer indicating the index of the iterated row. Both int1 and int2 are the same from what has been said before about the same index of row getting the origin_point from df_original and end_point from df_shift. Therefore we just care about the series (series1 from tuple1 and series2 from tuple2). We obtain these with *(_, original), (_, shift).* So original[FPLlat] returns the latitude of the origin_point from the row that we're iterating and *shift[FPLlat]* returns the latitude of the end_point.

Finally we reset the index of df_flightPlanExtended because otherwise it keeps the indexing from each concatenation, going from 0 to 50 and then starting again from 0 to 50, while what we want is that it goes from 0 to 600 (number of points).

Now with this expanded FlightPlan, we have a solid base with which we can assess the error calculations much easily. Recall that the idea of finding the best alienation and error in the trajectory is independent of time only after we have established a link that relates every point (or points) of the telemetry to a point of the FlightPlan.

To do this we need a way to find a relation between the two trajectories which are different lengths and that do not completely match considering the variations in speeds and deviations. We have seen it in previous figures when comparing the telemetry data to the flight plan data and the telemetry is not simply delayed but also scaled in the temporal axis in an irregular way, where some parts of the route are slower than intended while other parts take longer than intended. A way to do so without giving importance to these anomalies would be to either expand the FlightPlan dataframe to match the length of the telemetry and then match every point of the telemetry on a one-to-one relation. This is known as Euclidean Matching and it can be expressed in a simplified form as:

$$\boldsymbol{FlightPlan\ Trajectory\ X = x_1, x_2, \dots, x_i, \dots, x_n}$$

$$\boldsymbol{Telemerty\ Trajectory\ Y = y_1, y_2, \dots, y_i, \dots, y_n}$$

Where n is the number of points (rows) in the Telemetry file. The elements of each series are points in the format of $\boldsymbol{x_i} = (\boldsymbol{x_{i,lat}}, \boldsymbol{x_{i,lon}}, \boldsymbol{x_{i,alt}})$ and $y_{\boldsymbol{i}} = (y_{\boldsymbol{i},lat}, y_{\boldsymbol{i},lon}, y_{\boldsymbol{i},alt})$.

This idea in Euclidean Matching is to match point xi from the FlightPlan to point yi from the telemetry. For this to work both series need to have the same number of points. The FlightPlan could be interpolated so as to have the same time step as the telemetry and then match both time series by timestamp.

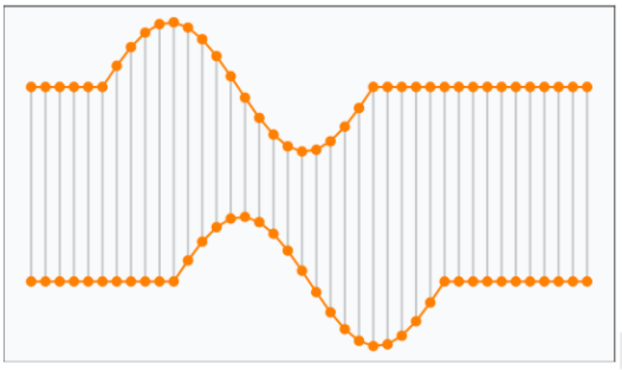


Figure 18. Euclidean matching between two time series, [8]

This is a straightforward approach that will yield the distance between these one-to-one pairs. However, $\boldsymbol{x_i}$ can be shifted in time in comparison to its pair y_i and this association would not output the actual adherence to the trajectory. In addition to that, both time series at our disposal have different lengths in terms of time. We can interpolate the FlightPlan as much as we want by adding points in between the trajectory but that does not change the duration of the FlightPlan. If the FlightPlan time, say t_{fp}, is the time at the last waypoint or in other words, the total time that it takes a drone to fly the intended trajectory, and t_{tel} the last telemetry time available, which is the time that it took the drone to follow the FlightPlan. In this situation, for a $t_{tel} > t_{fp}$, there is not a clear relation for the points and we have to find a solution to assess this problem.

This is where Dynamic Time Warping (DTW) comes in handy. Dynamic Time Warping is a technique used to compare and align two sequences of data that may have different lengths and variable speeds. It is commonly used in time series analysis and has various applications, including comparing trajectories, speech recognition, and gesture matching, [9].

To understand DTW, we can think about two sequences, like the flight plan and telemetry data. These sequences represent the movement of the drone over time. However, the sequences may not have the same length and may be distorted in terms of their timing. For example, as the telemetry data experiences delays and noise compared to the flight plan ideal route.

DTW aims to find the best alignment between these two sequences by warping and stretching their time axes. It allows for the comparison of corresponding points from both sequences, even if they occur at different times. This is done by finding a path through the sequences that minimizes the differences between the corresponding points. In other words, DTW finds the optimal way to match similar patterns in the sequences, even if they occur at different times or have different durations.

$$\textit{FlightPlan Trajectory } X = x_1, x_2, \dots, x_i, \dots, x_n$$
$$\textit{Telemerty Trajectory } Y = y_1, y_2, \dots, y_j, \dots, y_m$$

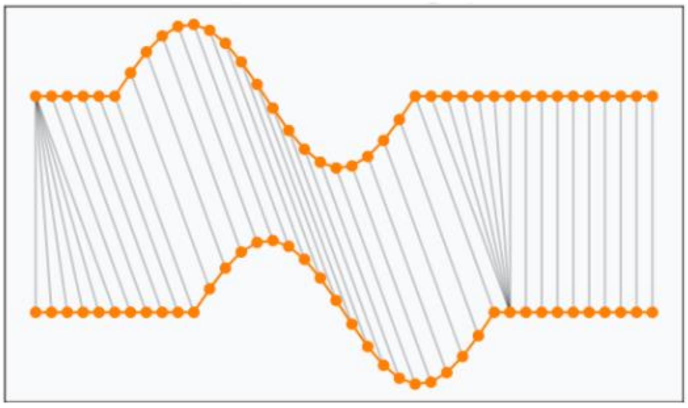
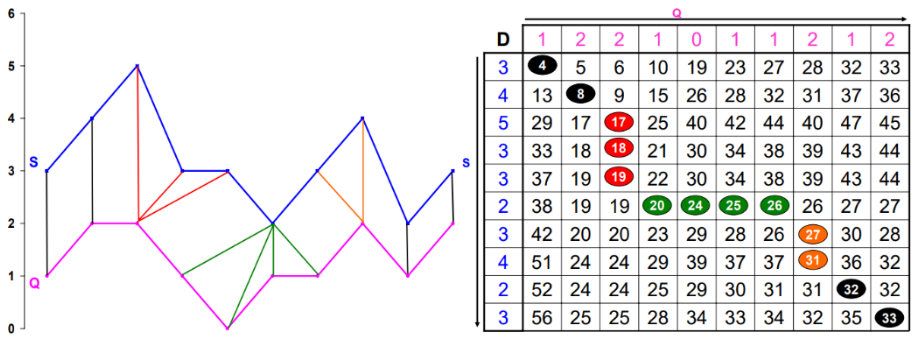


Figure 19. Dynamic Time Warping for two time series

Now, not every point is not necessarily matched to its corresponding point with the same index from the other time series and that the distinctive shapes of the trajectory are paired together. Nota that despite seeming that the distance of the lines linking both series are greater as they some of them are diagonal, we are establishing links between similar points and that this indicates both points x_i and y_j that are closer in space yet displaced temporally. A point of a time series can be paired with multiple points of the other time series as they have different lengths and all points must have a pair, closest in space, to compute the distance between them. This yields the best possible alignment, i.e. the minimum distance between time series.

$$D(i,j) = d(i,j) + min \begin{cases} D(i-1,j) \\ D(i-1,j-1) \\ D(i,j-1) \end{cases} \quad \textbf{(Eq. 7)}$$



D	1	2	2	1	0	1	1	2	1	2
3	4	5	6	10	19	23	27	28	32	33
4	13	8	9	15	26	28	32	31	37	36
5	29	17	17	25	40	42	44	40	47	45
3	33	18	18	21	30	34	38	39	43	44
3	37	19	19	22	30	34	38	39	43	44
2	38	19	19	20	24	25	26	26	27	27
3	42	20	20	23	29	28	26	27	30	28
4	51	24	24	29	39	37	37	31	36	32
2	52	24	24	25	29	30	31	31	32	32
3	56	25	25	28	34	33	34	32	35	33

Figure 20. Dynamic Time Warping sequence alignment and D matrix with optimal warping path, [10].

The optimal warp path is created based on the considerations of monotonicity, continuity and boundary. Monotonicity guarantees that the warping path does not roll back, in the sense that it will be always entire non-increasing or entirely non-decreasing (depending on how we arrange the axis). Continuity along the warping path makes sure that it only advances on step at a time. And the boundary condition guarantees that it the warping path contains all points of both series of data. For the implementation of DTW through code we have used the implemented function in the library tslearn.

Let's take this approach for some of our flights to visualize the alienation between the telemetry and FlightPlan. Recall that we are computing 3D distances between $x_i = (x_{i,lat}, x_{i,lon}, x_{i,alt})$ and $y_j = (y_{j,lat}, y_{j,lon}, y_{j,alt})$ and finding the optimal warping path.

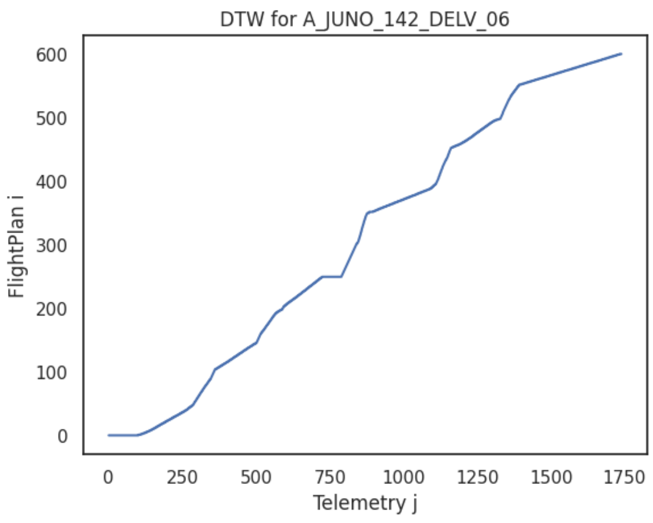


Figure 21. Example of warping path for one of the flights

We can represent the warping paths of all flights to visualize how is the general accuracy of the flights from out dataset. In Figure 22, the entire dataset has its DTW warping path plotted. The indexing is variable for all flights but the more straight the lines are, this is, without noise/irregularities, the better the adherence to the FlightPlan. Recall that every path starts at $(0, 0)$ and ends at (n, m).

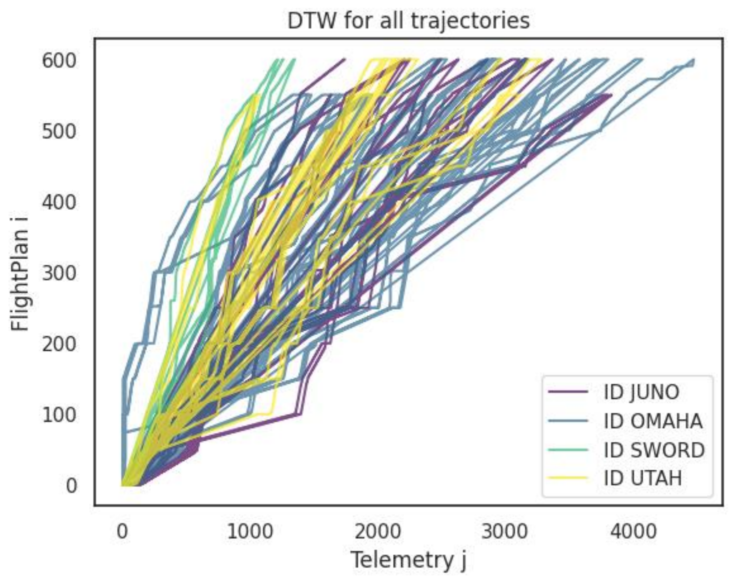


Figure 22. Warping paths for all the flights (colored based on the operator of the flight)

In addition, to get a more comprehensive look at what the DTW algorithm is doing, we can plot the distance association of a telemetry to a non-interpolated path, which computes the distance from the telemetry to the closest FlightPlan point. On the other hand, we can plot the DTW association between telemetry points and FlightPlan points. The plot is the view of a turn from the top and as the turn is not a perfect circumference, it is not trivial to tell which point of the telemetry related to the waypoint of the FlightPlan. It does not necessarily have to be the one with lowest distance but the one where the trajectories align better.

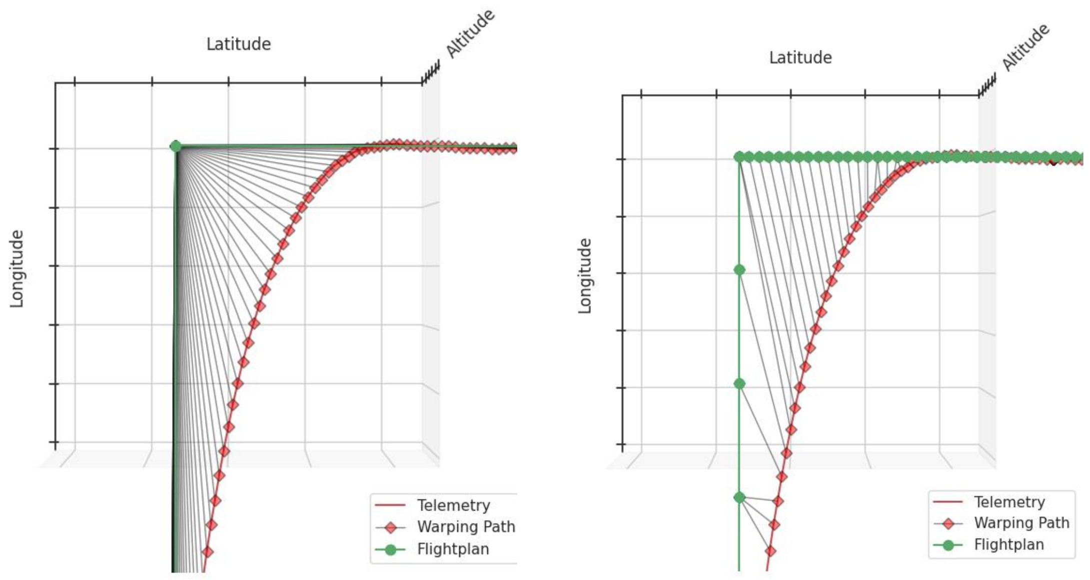


Figure 23. Dynamic Time Warping, no interpolation and interpolation

If we plot the whole trajectory it offers a better view of what the algorithm does. In this case, in Figure 24, we have chosen a flight whose telemetry has a notable error to better observe the DTW. Also, the number of interpolated points of the FlightPlan is highly reduced for visualization purposes.

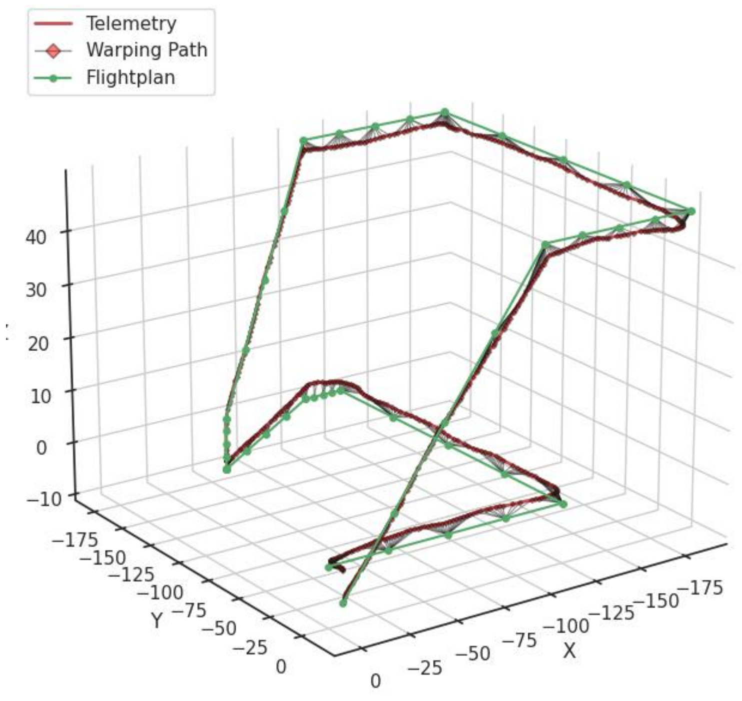


Figure 24. DTW algorithm linking telemetry points to FlightPlan (interpolated) points

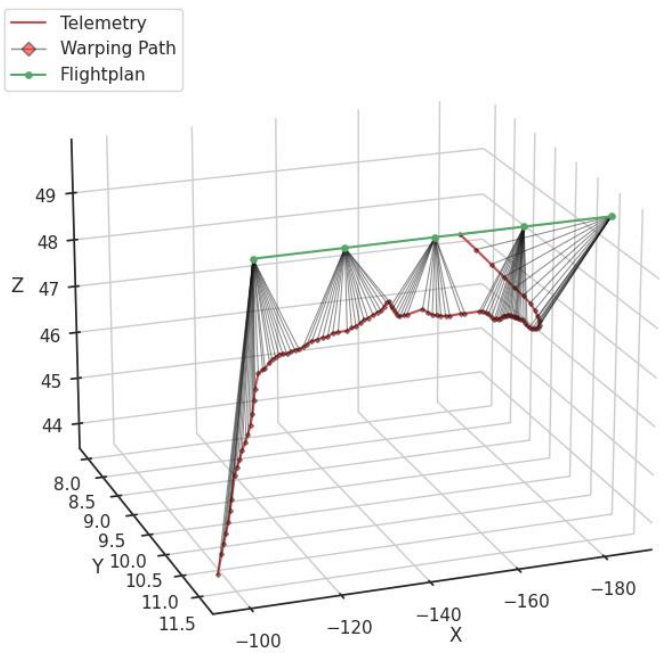


Figure 25. Close up the the association between FlightPlan points and Telemetry points using Dynamic Time Warping. The green line represents a single segment with 3 extra interpolated points for better results. This is simplified visualization purposed, the actual interpolation used of 50 points as computational resources allow it.

CHAPTER 3. MACHINE LEARNING

3.1. What is Machine Learning?

Machine learning is a branch of artificial intelligence (AI) that focuses on the development of algorithms and models capable of automatically learning and making predictions or decisions. In traditional programming, computers are given explicit instructions to perform specific tasks. However, machine learning takes a different approach by enabling computers to learn from data and examples, recognize patterns, and make act without being explicitly programmed.

The key idea behind machine learning is to design algorithms that can analyze and interpret data, identify meaningful patterns or relationships, and use them to generalize and make predictions on new unseen data. This ability to learn from data is what makes machine learning such a powerful tool. Instead of being programmed to perform specific tasks, machine learning algorithms learn from data through a process of training and adjustment of internal parameters.

During the machine learning process, relevant data is given as input. This data can be in various formats, such as images, text, or numerical values. We have to be conscious about what data is used to train the machine learning algorithm, and that is the reason of the data preprocessing step that is taken to clean, organize, and transform the data and ensure that the data is in a consistent format for the analysis. In our case we are working with numeric data and basic strings of text that will be converted to numeric values for simplicity.

The next step is training a machine learning model using the prepared data. The model is based on an appropriate algorithm that suits the problem that we need to solve. The algorithm is then provided with the prepared data, allowing it to learn and adjust its internal parameters with the patterns and relationships it discovers in the data. The learning process makes the algorithm manipulate the date iteratively, making predictions or decisions and comparing them to the correct outputs given also in the training data. When adjusting its parameters, the algorithm aims to minimize the difference between its predictions and the actual outputs.

Machine learning has different types of algorithms, including supervised learning, unsupervised learning, reinforcement learning and combinations between them. Each type has its own characteristics and applications. Supervised learning involves learning from labeled examples, where the algorithm is provided with input data along with corresponding target outputs. Unsupervised learning aims to find hidden patterns or structures in unlabeled data. And reinforcement learning involves learning through interactions and receiving feedback in the form of rewards or penalties. Briefly, the models are trained with a partition of the data, called train data.

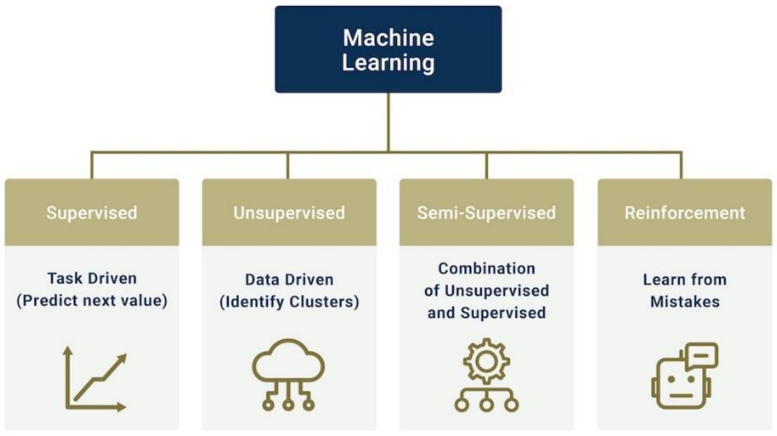

Machine Learning
Supervised
Task Driven (Predict next value)
Unsupervised
Data Driven (Identify Clusters)
Semi-Supervised
Combination of Unsupervised and Supervised
Reinforcement
Learn from Mistakes

Figure 26. Machine Leaning types, [11]

3.2. Main machine learning types

In order to put the data to the test a specific machine learn model has to be chosen. Models differ in the way that they learn from the data. The way the algorithms work for each model is what makes them different from one another and their applications are distinct. Simpler models like Linear Regression, that will be explained later, use linear equations to predict the outputs while other more advanced models such as Random Forest are based on hierarchical decision and rules to observe the patterns in the data.

3.2.1. Supervised Learning

Supervised machine learning is a type of machine learning where the algorithm learns from labeled training data. In supervised learning, the input data (features) and the corresponding output labels are provided to the algorithm during the training phase. The goal is to learn a mapping function that can accurately predict the output labels for unseen input data. Therefore, this is clearly our case as we will provide both the input and output to train the model and after the training phase, we will just provide the input and obtain the predicted output.

The model finds the patterns with the output labels and then it is tested with the other partition of the data called test data. From the test data we get the evaluation metrics as it is unseen data.

3.2.2. Unsupervised Learning

Unsupervised machine learning is a type of machine learning where the algorithm learns patterns and structures in unlabeled data. Unlike supervised learning, there are no output labels provided in unsupervised learning. The goal is to discover hidden patterns, relationships, or clusters in the data.

3.3. Machine Learning Models

3.3.1. Regression Models

Regression models are used when the target variable is continuous, and the goal is to predict a numeric value. In the given list of models, the following models fall under this category:

3.3.1.1. Linear Regression:

Linear regression assumes a linear relationship between the input features and the target variable. It fits a linear equation to the data by minimizing the sum of squared differences between the observed and predicted values.

3.3.1.2. Random Forest

Random Forest is an ensemble of decision trees. It combines multiple decision trees to make predictions and provides an average prediction based on the predictions of individual trees.

3.3.1.3. Gradient Boosting Regressor

Gradient Boosting Regressor also combines multiple decision trees, but in a sequential manner. It fits each subsequent tree to the residuals of the previous tree, improving the predictions gradually.

3.3.1.4. Extreme Gradient Boosting (XGBoost)

XGBoost is an optimized implementation of gradient boosting that provides faster and more accurate predictions. It incorporates regularization techniques and advanced algorithms to boost the performance. Showed good performance in [12] in a similar drone application.

3.3.1.5. Light Gradient Boosting (LightGBM)

LightGBM is another gradient boosting framework that aims for faster training speed and lower memory usage. It uses a novel tree-growing algorithm and various optimization techniques.

3.3.2. Classification Models

Classification models are used when the target variable is categorical, and the goal is to assign the input data to one of the predefined classes. In the given list, the following models fall under this category:

3.3.2.1. Logistic Regression

Logistic regression is a binary classification algorithm that models the probability of an instance belonging to a certain class. It uses a logistic function to map the input features to the target class probabilities.

3.3.2.2. Naive Bayes

Naive Bayes is a probabilistic classification algorithm based on Bayes' theorem. It assumes that the features are conditionally independent given the class label and calculates the probabilities of each class based on this assumption.

3.3.2.3. Support Vector Regression

Support Vector Regression (SVR) is typically used for regression tasks, but it can also be adapted for classification. SVR maps the input features to a higher-dimensional space and finds a hyperplane that maximizes the margin between the classes.

3.3.2.4. Instance-based Learning Models

Instance-based learning models make predictions based on the similarity between instances in the training data and the test data. In the given list, the K-Nearest Neighbors (KNN) algorithm falls under this category. KNN classifies or regresses a new instance by finding the K closest instances in the training data and predicting based on their labels or values.

3.4. Detecting outliers in the data

When it comes to training a machine learning model, the approach we take on data input plays a crucial role in its performance. In our specific case, we have from the files a decent amount of data that could potentially be relevant for training the model. However, it is important to be cautious and not saturate the model with an excessive amount of data in the hope of achieving the best performance. As we have discussed, in traditional programming, specific behaviors can be defined, machine learning models define these themselves and despite some behaviors may be apparent to us, they might not be to the model if the data is not well provided. Therefore, we need to make decisions when it comes to choosing the most relevant and informative input data for the model and choosing as well an output variable (or variables) that the model will try to predict.

At the same time, even with the right amount of input variables, we have to make sure that the data which we are providing is not corrupt or contain values that are abnormal in comparison to the rest of the dataset. This could be for several reasons but in the case of our data, it may be the case with the acquisition of the data or when writing it.

As we are dealing with positions in time mainly, we have seen in Figure 7 that some of the telemetry files have failed at completing the trajectory and the data regarding the part of the trajectory that is outside the typical functioning of the drone can't be considered for the analysis as it would skew the results and reduce performance.

We can make use of the Dynamic Time Warping algorithm that we have created in Chapter 2. This will come in very handy to identify trajectories whose telemetry has not adhered to the FlightPlan. It can be done by analyzing the error during the trajectory, [13]. When the drone deviates from the trajectory, the warping path also deviates from the diagonal line which represents the ideal adherence and the more it deviates the higher the error and the odds of being an abnormal flight. In Figure 27 we can visualize both the maximum error of the flights in meters as well as the average errors. Both graphs show similarities but we specially take into account the average errors as maximum errors can likely associated to the drone telemetry data before it even reaches the start of the route or continuing to record data after the trajectory. In this case, the average error accounts for that and it will remain low during the trajectory to still be considered.

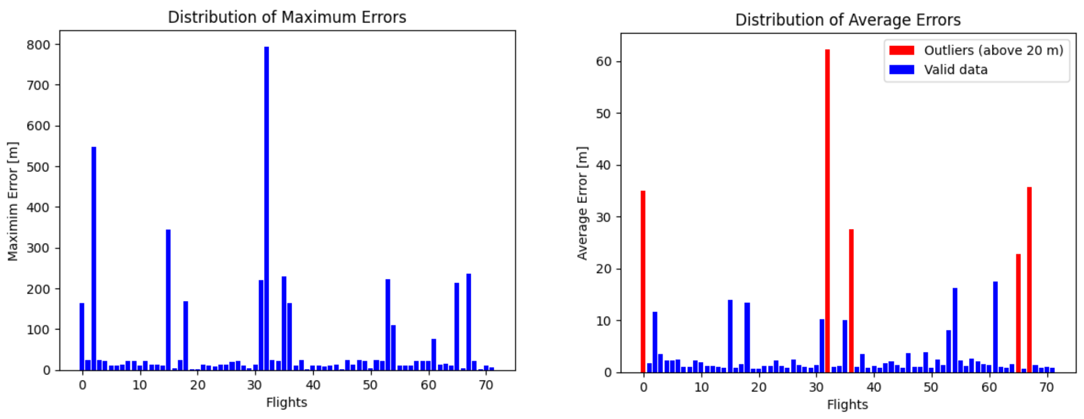


Figure 27. Error distributions for time independent analysis

Some of the flights have only recorded a part of their telemetry as can be seen in Figure 27. These flights have a high maximum error but the average is remarkably low yet still affected by the maximum error values. The case seen below represents the third flight seen in Figure 27, the one on the left with maximum error around 550 m and we can see that its average is within the 20 m threshold. The value of 20 m is chosen based on inspecting the flights with higher error and seeing that above 20 meters the telemetry is highly unrelated to the FlightPlan, while the ones below 20 m but with relatively high error represent flights with missing data, see Annex to have a closer look on these flights. We will not use the part of the telemetry and FlightPlan that we don't have from and later we will analyze by segments which will examine segments individually instead of the whole trajectory, resulting in lower average errors in cases like this one.

This discussed method of detecting the outliers is not used to generate input data for the model. The error values could be used if necessary but these will not be considered as a feature in the model data. After this procedure the data is cleaner and slightly reduced. As we do not know why the data is missing or the why the drone did not adhere to the intended trajectory, it is questionable whether to consider the data where the drone does adhere to the telemetry because it may have experienced some flying conditions out of the usual which led the drone to not properly record the data.

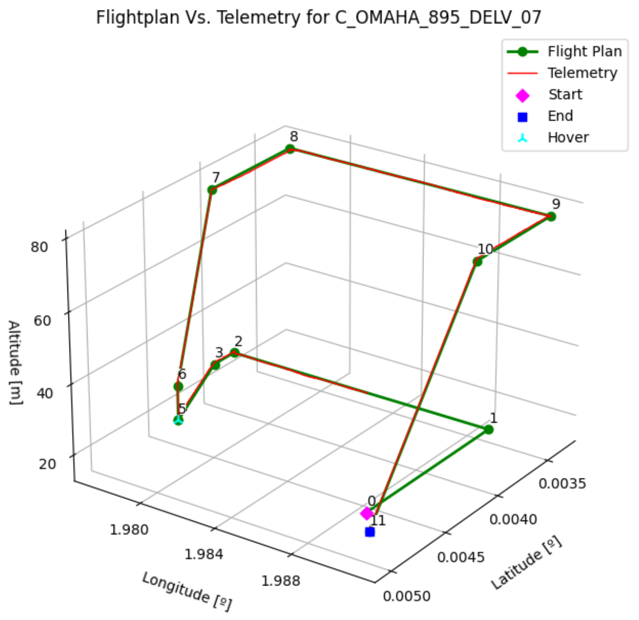


Figure 28. Example of a telemetry file with missing data

3.5. Machine Learning dataset

Let’s discuss the different features that can be obtain from the data that we have and that can potentially be useful for the model. These have to include information about the flight that affects the adherence to the trajectory. Depending on what we want to predict as an output, the importance of the different input data be altered. This is, if we want to predict time-related parameters, the most beneficial inputs will be likely time-related yet that does not exclude other features may also play an important role.

In addition to that, as machine learning is based on learning patterns, for it to be as accurate as possible, it is essential that the input data exhibits some sort of patterns across the dataset. In the case of our dataset, we count with tenths of flights that, despite all of them being different, they hold similarities and the model is able to establish connections between behaviors happening in a route and see great results when training in another route. If we were to have a dataset whose flights had no similarities in terms of the segments, duration and speed, basically being all of them unrelated, the odds of the results being less accurate would be high.

3.5.1. Features

3.5.1.1. Spatial features

The first feature that is considered for the flights and that can be considered is the positional values of the trajectory. Both the FlightPlan and the telemetry contain valuable information of the points in space, however, it is not a single point that matters but the relationship of two consecutive points. Providing a single point per line in a dataframe can be irrelevant as there is no association that can be done with another point for that particular entry of data. That is why working with segments has potential benefits as the relation between x, y and z magnitudes has a strong effect on the prediction of the model as it contains essentially the attributes of that segment.

While it is true that considering the origin point of a segment as (0, 0, 0) and the giving simply the corresponding value to the end point might seem like an easier approach for the model as it deals with less data. However, considering what we have mentioned earlier about the similarities and relationships between routes, it comes in handy too to give information about the starting point too as the performance of the drone can be different based on the altitude or in certain areas of a given trajectory. Note that for this latter assumption, ECEF coordinates for the raw points have to be used as to identify particular patterns based on the location of the drone. ENU coordinates depend on the reference of each flight which is set to the origin point of the FlightPlan. Unless the reference point is the same for all flights, the ENU coordinates contain information useful for every individual flight yet we cannot generalize with merely the information about position.

3.5.1.2. Temporal features

Time also plays a critical role in terms of predictions as it contains valuable information about the intended position of the drone along the trajectory. In Chaper 2 we have seen a way to compute the theoretical time that it should take a drone to fly the FlightPlan trajectory. This is an obtained feature from the raw data and it can be extremely helpful when compared with the telemetry time as a way to get the delay that the drone experiences in every segment.

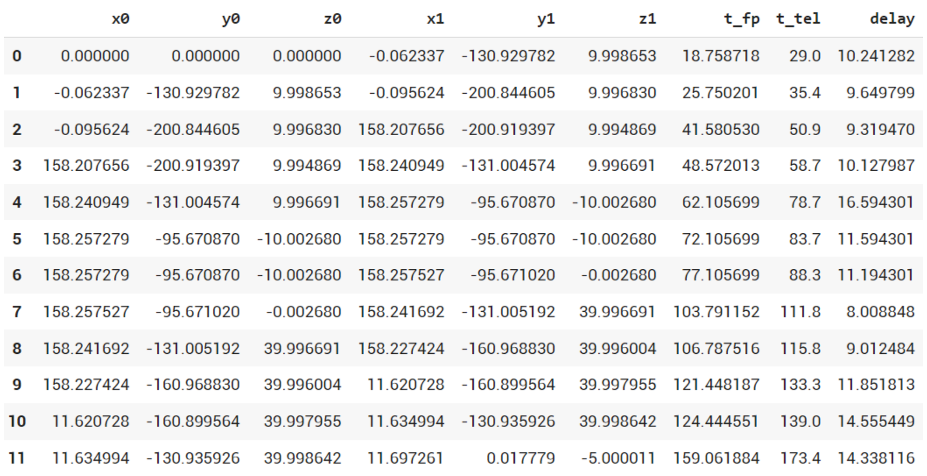
When generating the time data for the model, we want to associate a telemetry time to each segment, for instance the time at the end of the segment for both telemetry and FlightPlan, the difference between these times is the delay in that given segment. If necessary, we can use $delay = t_{tel} - t_{fp}$ where t_{tel} is the time at of the telemetry at given point and t_{fp} the time of the FlightPlan at that point. Finding the t_{tel} for a given point can be expensive in terms of computation resources as telemetry points rarely coincide exactly with the FlightPlan points and the approach of computing the distance from the desired point to all the points of the telemetry to find which one is closer is not a reliable method as the drone may be closer to another segment during that particular time, specially is segments were to be very close, separated a distance of around the average error, would result in wrong associations between the times.

To obtain the telemetry time associated with a specific point in the Flight Plan, we use the power of Dynamic Time Warping (DTW) once again. As we have commented earlier, DTW establishes a link between all points in the trajectory, allowing us to effectively map the Flight Plan points to their corresponding telemetry points. This wat, when identifying a point in the Flight Plan, we can effortlessly locate its associated telemetry point using the DTW alignment as it ensures that each point in the FlightPlan is properly matched with its corresponding point (or points) in the telemetry data. Once we have identified the telemetry point linked to the desired Flight Plan point, we can obtain the associated time value stored in the corresponding row of the telemetry data. In a similar manner we can use this to obtain the telemetry position at that time if the objective is to predict positions.

In Listing 3 we can see the way this time is obtained. Essentially, we are looking for the associated index to a particular FlightPlan segment in the synchronized dataframe that contains the correlation between the segments. Knowing the telemetry index we can easily get its time.

```
  for i, row in model_df.iterrows():
    model_df.loc[i, "t_tel"] = telemetry.iloc[sync[np.where(np.array(sync)[:,1] ==
find_nearest(np.array(sync)[:,1],(i+1)*n))[0][0]][0]]['secs']
```

Listing 3. Closest telemetry time obtention



	x0	y0	z0	x1	y1	z1	t_fp	t_tel	delay
0	0.000000	0.000000	0.000000	-0.062337	-130.929782	9.998653	18.758718	29.0	10.241282
1	-0.062337	-130.929782	9.998653	-0.095624	-200.844605	9.996830	25.750201	35.4	9.649799
2	-0.095624	-200.844605	9.996830	158.207656	-200.919397	9.994869	41.580530	50.9	9.319470
3	158.207656	-200.919397	9.994869	158.240949	-131.004574	9.996691	48.572013	58.7	10.127987
4	158.240949	-131.004574	9.996691	158.257279	-95.670870	-10.002680	62.105699	78.7	16.594301
5	158.257279	-95.670870	-10.002680	158.257279	-95.670870	-10.002680	72.105699	83.7	11.594301
6	158.257279	-95.670870	-10.002680	158.257527	-95.671020	-0.002680	77.105699	88.3	11.194301
7	158.257527	-95.671020	-0.002680	158.241692	-131.005192	39.996691	103.791152	111.8	8.008848
8	158.241692	-131.005192	39.996691	158.227424	-160.968830	39.996004	106.787516	115.8	9.012484
9	158.227424	-160.968830	39.996004	11.620728	-160.899564	39.997955	121.448187	133.3	11.851813
10	11.620728	-160.899564	39.997955	11.634994	-130.935926	39.998642	124.444551	139.0	14.555449
11	11.634994	-130.935926	39.998642	11.697261	0.017779	-5.000011	159.061884	173.4	14.338116

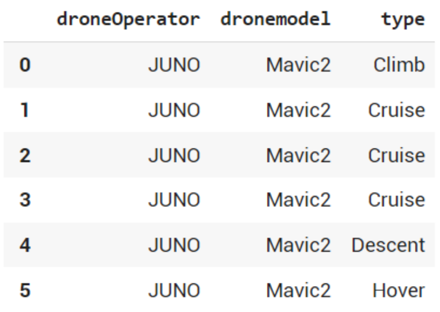
Figure 29. Dataframe with examples of temporal features

3.5.1.3. Drone characteristic features

Information that we also have available which is not initially in a numerical value is the drone model type. The drone model can have an impact on the adherence to the FlightPlan as higher end drones count with better specifications which can potentially contribute to a better awareness and therefore better response to the FlightPlan guidelines. At first glance we are not going to analyze by model type but this data can be provided to the model for its algorithms.

Other comparable features include the segment type, which can either be: descent, climb, cruise and hover. Knowing this data is helpful as drones can have different performances depending on the kind of segment that they are dealing with.

In addition, the operator of the drone is a factor to take into account as operators may use different criteria and software when setting up the drones. This however, is a piece of information that we do not know for sure and we will let the machine learning model assess the data and find the patterns, if there are any.



	droneOperator	dronemodel	type
0	JUNO	Mavic2	Climb
1	JUNO	Mavic2	Cruise
2	JUNO	Mavic2	Cruise
3	JUNO	Mavic2	Cruise
4	JUNO	Mavic2	Descent
5	JUNO	Mavic2	Hover

Figure 30. String encoded values for a flight

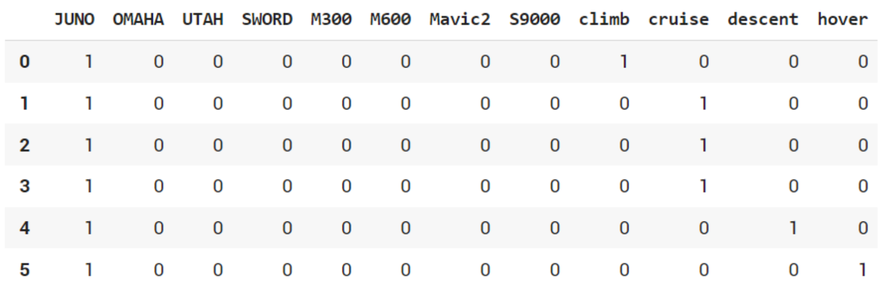
For machine learning purposes, the general tendency is to acquire as much data as possible in order to capture a wide range of variations and patterns related to the problem to solve. However, it is important to note that handling an excessively large dataset can also impact computation time and resource requirements. Therefore, it becomes essential to find the optimal trade-off that suits a specific scenario.

In our case, for the drone flights, we observe that they follow specific airways with consistent altitudes, and this pattern is repeated across multiple flights and delivery scenarios. Additionally, we notice a consistent sequence of actions, such as starting with a climb, followed by a hover at some point of the trajectory, and ending with a descent. These patterns are helpful and justify the feasibility of solving our problem. Having a dataset that shows such patterns is essential for developing an accurate model. Random flights performed by unknown operators with irregular routes would lack the necessary correlation for the machine learning model to properly identify and understand the provided data. Without the right dataset, the predictions derived from the model would likely be inaccurate and unreliable. However, we are not going to limit the model with any of these conditions, as the model is suitable to be trained with data that showcases some sort of patterns, yet we will not generalize and make assumptions for all flights, the model will be able to do this on its own.

3.5.2. One Hot Encoding

For the model to understand the string values that we will provide, it needs a numerical value that can be used for the pattern recognition. A possible option is to replace the string values by numbers representing each case while keeping the values in the same column such as representing the model type as number 0 for climb segments, number 1 for cruise and so on. This method is a simple way to assign numeric values to different categories of a categorical variable. However, it has a significant limitation in that these numeric values can be misinterpreted by some machine learning algorithms. For example, if we encode four drone models with values 0, 1, 2, and 3, an algorithm may incorrectly interpret that the operator corresponding to value 3 is three times greater than the drone model with value 1, which is not true.

An alternative to this method is the method called One Hot Encoding. This strategy involves creating a binary column (which can only contain values 0 or 1) for each unique value in the categorical variable being encoded. The column corresponding to the current value in each row is marked with a 1, while the remaining columns are assigned a value of 0. For example, in the case of the drone operator variable of the dataframe, One Hot Encoding creates four binary columns (JUNO, OMAHA, UTAH and SWORD). For each flight, a value of 1 is assigned to the column corresponding to their operator, and a value of 0 is be assigned to the columns of the other operators. This way, each flight is represented by a binary vector indicating the presence or absence of each categorical value, which avoids the possibility of the model and algorithms to misinterpreting the numeric values assigned by other ways of encoding.



	JUNO	OMAHA	UTAH	SWORD	M300	M600	Mavic2	S9000	climb	cruise	descent	hover
0	1	0	0	0	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	0	0	0	1	0	0
2	1	0	0	0	0	0	0	0	0	1	0	0
3	1	0	0	0	0	0	0	0	0	1	0	0
4	1	0	0	0	0	0	0	0	0	0	1	0
5	1	0	0	0	0	0	0	0	0	0	0	1

Figure 31. One hot encoded string labels

3.5.3. Metrics for evaluation

R2 (R-squared), MSE (Mean Squared Error), and MAE (Mean Absolute Error) are commonly used metrics to evaluate the performance of machine learning models. They provide insights into how well the model fits the data and how accurate its predictions are.

R2, is a statistical metric used to evaluate the how good the model fits the data for a regression model. It provides an indication of how well the independent variables (features) in the model explain the variability in the dependent variable (target). It ranges from 0 to 1, with 1 representing a perfect fit where the model predicts all the variability in the target variable, and 0 indicating that the model does not have any precision in the predictions it makes. It measures the proportion of the variance in the target variable.

MSE is a commonly used metric to assess the accuracy of models. It calculates the average of the squared differences between the predicted values and the actual values of the target variable. Squaring the differences makes MSE yield higher weight on larger errors. It penalizes the model for larger deviations between predicted and actual values. MSE is useful for evaluating the overall quality of predictions and provides a measure of the average squared deviation between the predicted and actual values.

MAE is another metric used to evaluate the performance of regression models. It calculates the average of the absolute differences between the predicted values and the actual values of the target variable. In this case, it does not square the errors. It takes the absolute value of the differences. This means that all errors are weighted equally, regardless of their direction. It provides a more interpretable measure of error, as it represents the average magnitude of the errors in the original units of the target variable.

CHAPTER 4. Results

Having at our disposal the features that we have computed, the next stage is training and testing the models. Machine Learning does require trial and error to find the optimal features which yield the best performance. The aim is to test the machines learning models discussed in the previous chapter and evaluating their performance. The results with the best performance will be tested further with validation data.

As the last step the obtained machine results will be compared to the U-Space predictions and to the real telemetry data. This way, it will be possible to determine if we have made a positive contribution to U-Space by having more accurate predictions.

4.1. Time predictions

The prediction of time at specific points along the FlightPlan trajectory is a significant aspect of our analysis. To achieve this, we provide segment-related input data, where each row contains relevant information about the segment as well as characteristic details about the drone. Our objective is to predict the corresponding telemetry time, also denoted as y (output variable), which we have previously computed with the help the telemetry files. This telemetry time represents the moment when the drone is closest to the designated FlightPlan point, taking into account the nature of the trajectory. It is crucial to ensure that the predicted time corresponds accurately to the intended point along the FlightPlan and is not mistakenly associated with a point from a later segment of the trajectory that might be closer.

For the time predictions, we will train the model with segments of all flights, all of them as independent rows in the dataframe. The intention is to be able to split every flight into segments so that in the end we have a database with all segments of all routes, without any relation to one another. All the origins of the individual segments are brought to the origin at coordinates (0, 0, 0) and they end at their respective points. The time at the start of each segment is set to 0 seconds and it also has the end time as computed in the mathematical approach of Chapter 2 and its related telemetry time. In addition, there is information of the drone, operator that flew this segment and the segment type (yet this latter one can likely be deduced by the model from taking int consideration the information about the origin and end points).

Input data: *['JUNO', 'OMAHA','UTAH', 'SWORD', 'M300', 'M600', 'Mavic2', 'S9000', 'z0', 'x3', 'y3', 'z3', 'vel_fp', 'turn_fp', 't_seg_fp','3Ddist']*

Where *['JUNO', 'OMAHA', 'UTAH', 'SWORD'], ['M300', 'M600', 'Mavic2', 'S9000']* are the one hot encoded columns of ‘dronemodel’ and ‘droneOperator’ respectively.

Let's discuss the reason behind this selection of columns, which from the computed features, appears to be the selection of features that performs better after testing the possible combinations.

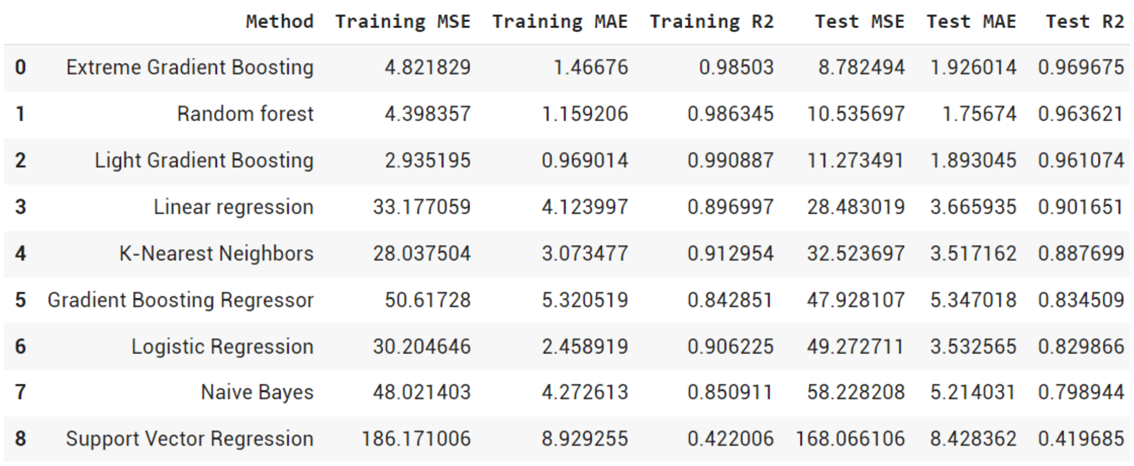
We include $x3, y3, z3$ as they represent the point in space where, from (0, 0), it creates the line making a the segment. This way we avoid using origins and end points that the model could misinterpret or memorize. Also $z0$ is altitude at which the segment starts from as it can help the model identify patterns related to airways, that are in constant altitudes.

In addition, we provide parameters about the segments such as $vel\_fp, turn\_fp$ that represent the velocity and turn radius specified by the FlightPlan. Finally, we provide the theoretical segment time, t_seg_fp, and 3D distance of the segment, $3D\ dist$.

The output data is the telemetry time, variable y: *t_seg_tel*

In this model, the dataset contains 690 rows which represent individual segments. The excluded flights are only the outliers, this is, 5 flights. The flights with missing data can be included in exception of the segments where the data is missing as we are considering individual segments.

The 9 models are trained with the same exact data and the obtained results are shown in Figure 32.



	Method	Training MSE	Training MAE	Training R2	Test MSE	Test MAE	Test R2
0	Extreme Gradient Boosting	4.821829	1.46676	0.98503	8.782494	1.926014	0.969675
1	Random forest	4.398357	1.159206	0.986345	10.535697	1.75674	0.963621
2	Light Gradient Boosting	2.935195	0.969014	0.990887	11.273491	1.893045	0.961074
3	Linear regression	33.177059	4.123997	0.896997	28.483019	3.665935	0.901651
4	K-Nearest Neighbors	28.037504	3.073477	0.912954	32.523697	3.517162	0.887699
5	Gradient Boosting Regressor	50.61728	5.320519	0.842851	47.928107	5.347018	0.834509
6	Logistic Regression	30.204646	2.458919	0.906225	49.272711	3.532565	0.829866
7	Naive Bayes	48.021403	4.272613	0.850911	58.228208	5.214031	0.798944
8	Support Vector Regression	186.171006	8.929255	0.422006	168.066106	8.428362	0.419685

Figure 32. Results obtained from the models (R2 sorted)

The evaluation of the metrics suggests that the XGB model performs exceptionally well in terms of time predictions. It is true that it presents the lowest MSE, as well as the highest R-squared, however it is important to emphasize before concluding, that it might still be worth to keep its close competitors. As these metrics are obtained from the test data used during training, they do not strictly guarantee in every situation that the model will outperform its closest competitors, such as the Random Forest and LGB models as the difference between their metrics is very slight in this case.

The subtle differences definitely guarantee these top scoring models will have remarkable prediction capabilities, however their performance on validation data might not maintain the same order as in the previous figure. It is important to consider the context and possible variations in performance when providing new and unseen data. Therefore, even though the XGB model may show the best performance with the evaluation metrics, we can still need to verify and observe its behavior with validation data.

For the validation data, in our case full trajectories, we take the four top scoring models and test them once again with this data that we can actually visualize as we predict the time for a whole trajectory and not random segments like it is done with the test data.

The following Figure 33 shows the results of the validation data which are of 5 flights, which have been checked that they are not outliers or contain missing data so we can compare them smoothly with the entirety of the telemetry.

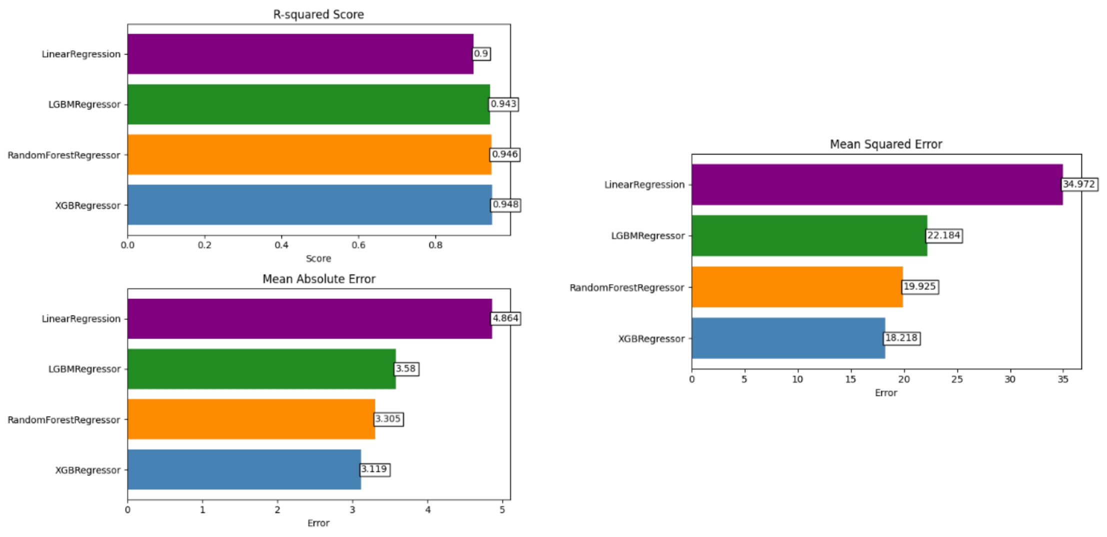
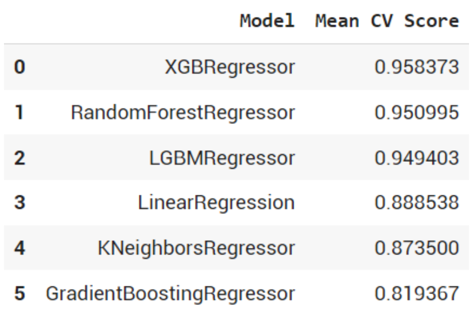


Figure 33. Metrics for the 4 best performing models (with validation data)

The XGB regressor has the best performance even for the MAE, with the testing done with validation data. Note that the smaller the MAE and MSE the better while for the R2 the higher the better.

And finally, to further understand which model performs better for this time analysis, we can perform cross validation so the models can be trained and tested with all the data, except from the validation data. The cross validation score is the average of the scores obtained from each data split while choosing a different fold of test data every time. We expect the difference to be small as generally a test/train split selecting random pieces of data results in a non-biased dataframe, which is what we want. As it can be seen, the 6 best performing models are represented in the dataframe below and the ranking remains the same.

We could take this step before training the models and merely pick the top one scorer to fit it with the data.



	Model	Mean CV Score
0	XGBRegressor	0.958373
1	RandomForestRegressor	0.950995
2	LGBMRegressor	0.949403
3	LinearRegression	0.888538
4	KNeighborsRegressor	0.873500
5	GradientBoostingRegressor	0.819367

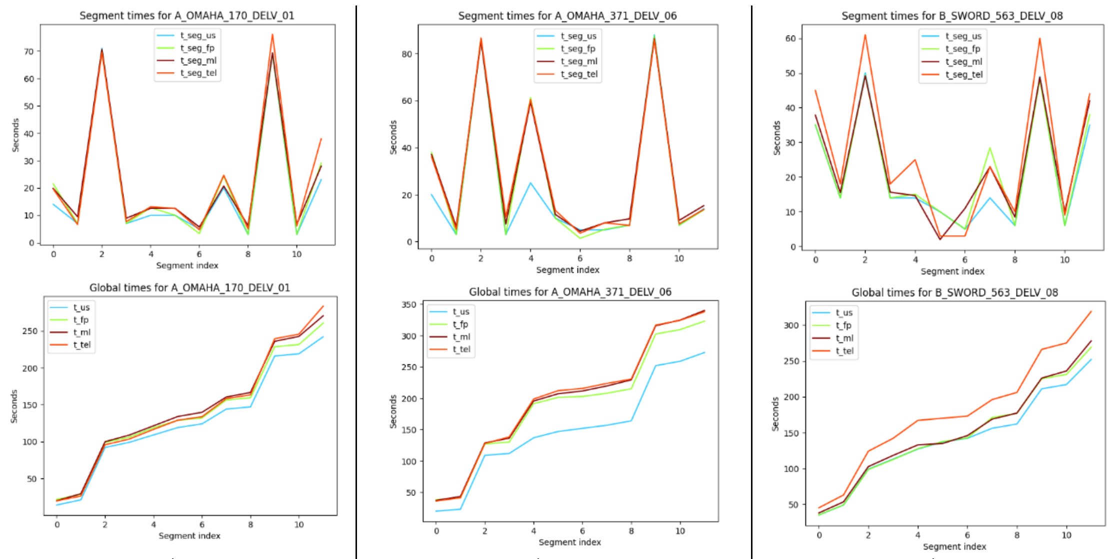
Figure 34. Cross Validation (CV) scores using k-fold validation.

Now we can plot the results of the validation data for some trajectories and see how the model performs. We will choose the XGB regressor as it showed the best performance and accuracy.

In the plots, four times will be represented:

- $t\_seg\_fp \rightarrow$ for the theoretical time of every segment of the FlightPlan as computed in Listing 1.
- $t\_seg\_us \rightarrow$ for the time of each segment according to the U-Space prediction.
- $t\_seg\_tel \rightarrow$ for the telemetry time, where every segment has been delimited with the use of Dynamic Time Warping (see section Time Independent Error Analysis)
- $t\_seg\_ml \rightarrow$ for the machine learning model predicted time of each segment. In this case, using the XGB model.

We can plot the time per segment individually and then the accumulated time as the representation of the increasing time during the trajectory.



Time	MSE
t_us	315.8
t_fp	74.7
t_ml	28.9

Time	MSE
t_us	2943.57
t_fp	127.96
t_ml	7.07

Time	MSE
t_us	1652.2
t_fp	1078.7
t_ml	578.5

Figure 35. Upper row: segment time predictions ($\boldsymbol{t\_seg\_ml}$). Added reference times for comparison $\boldsymbol{t\_seg\_fp}$, $\boldsymbol{t\_seg\_us}$, $\boldsymbol{t\_seg\_tel}$; Middle row: global accumalated trajectory times; Bottom row: MSE for each of the times with respect to the ground truth, which are telemetry related times

A more intuitive way of visualizing the prediction of times is by plotting the coordinates over time. During this project, we have seen various representations of telemetry and FlightPlan data over time. Now, we can take the results of our predictions and plot the FlightPlan waypoints at the predicted cumulative sum of t_seg_ml (t_ml). This allows us to observe how the original FlightPlan will look like with the predicted times, resembling the telemetry data. In Figure 36 we plot also the input of to the model. Essentially, the lines labeled as FP are the representation of the FlightPlan coordinates with the mathematically computed t_fp. The dashed line labeled as TEL represents the actual flown trajectory, the telemetry coordinates over time. And the lines labeled as ML represent the FlightPlan coordinates at the predicted times.

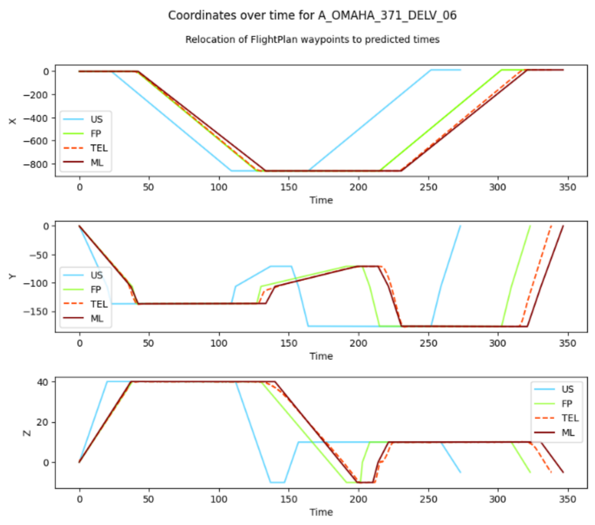


Figure 36. Coordinates over time. Spatial locations (X, Y, Z) of the predicted lines correspond to the input coordinates that have been relocated to the predicted times with the XGB model.

In other cases, it is not as accurate. However, as it can be seen in the MSE, the error is still lower than the U-Space approach and mathematical approach.

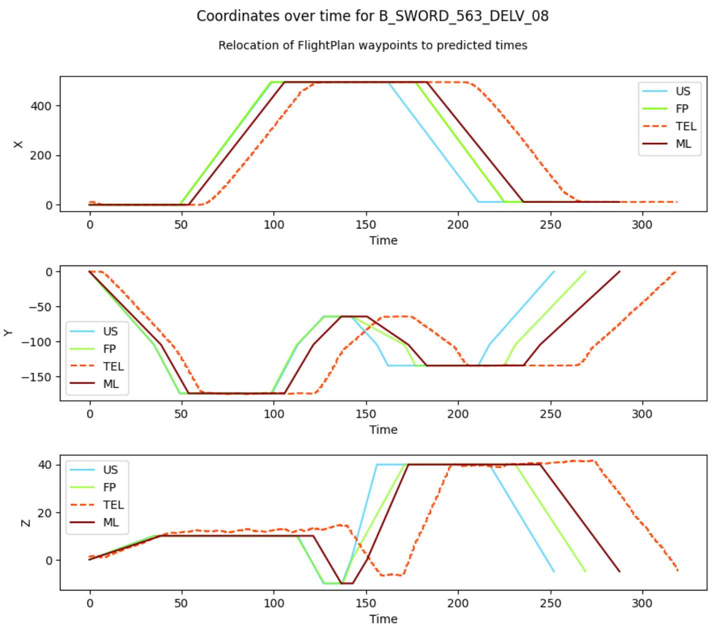


Figure 37. Another, not as accurate plot of coordinates over time. Spatial locations (X, Y, Z) of the predicted lines correspond to the input coordinates that have been relocated to the predicted times with the XGB model.

We can also get a more detailed view of this models' performance by plotting the feature importances. Models utilize the features differently and below we see two example of the importance given to each feature (note that the Y scale is not the same). At first sight it appears that LGB is making a better use of the features, however that does not necessarily mean that the predictions are better.

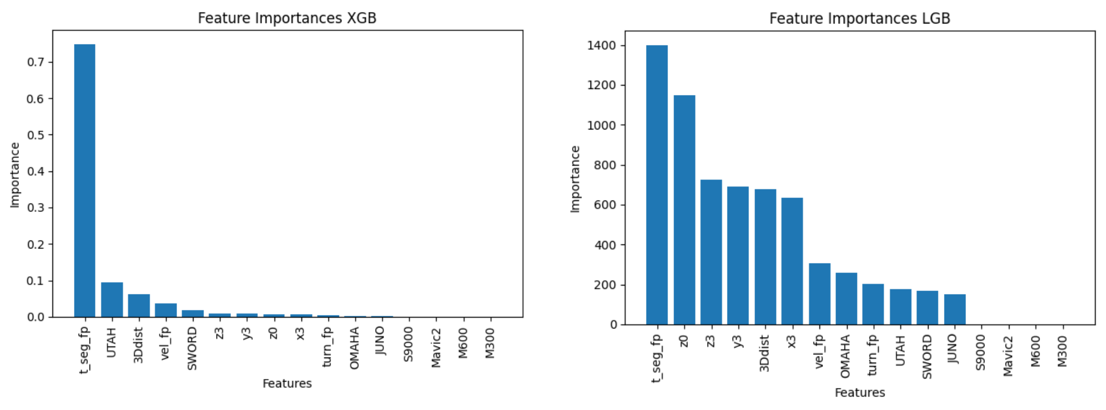


Figure 38. Feature importances for the XGB model (left side figure) and for the LGB model (right side figure)

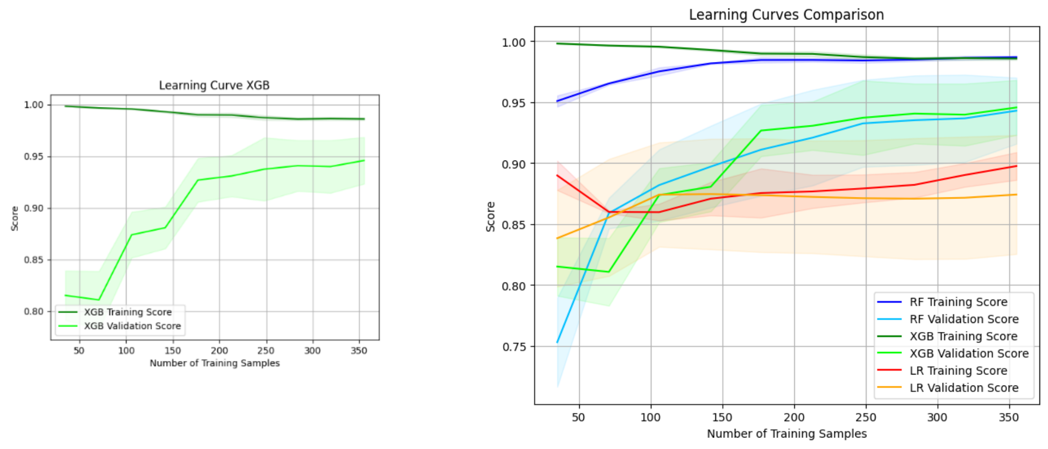


Figure 39. Left side plot: Learning Curve for the XGB model; Right side plot: Comparison of multiple learning curves (XGB included)

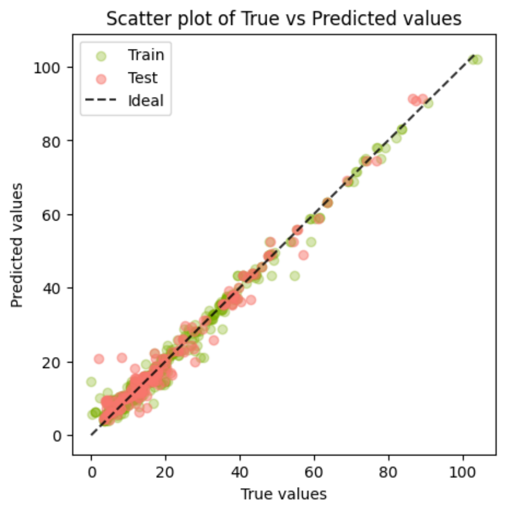


Figure 40. Scatter plot of True (t_tel) and predicted values (t_ml) for the XGB Regressor model. Points on top of the ideal dashed line indicates perfect predictions.

4.2. Position predictions

When it comes to predicting positions, it's important to note that the features obtained from the available data may have a weaker performance in the final model. This is mainly due to the fact that telemetry positions (or related spatial metrics), for instance, at the end of a segment, are extremely variable and the noise in this kind of data will make it tough for a model to predict it accurately.

In contrast, the prediction of time benefits from more suitable features that we obtain in the data. Features such as distance, velocity, and segment properties play a significant role in determining the time required to fly a particular segment. For instance, we include a time reference feature (t_fp) which is particularly influential, as it closely related with the ground truth time (t_tel). Even without t_fp, the relationship between distance and velocity yields that the predicted value is typically similar or higher than t_fp. And this relationship is highly consistent, as we have observed from the data.

However, when it comes to predicting the position of the drone, these features do not necessarily help determine the position at a given time. While time prediction benefits from a strong relationship between feature and the target variable, predicting positions requires also strong relationships that we might not be able to obtain.

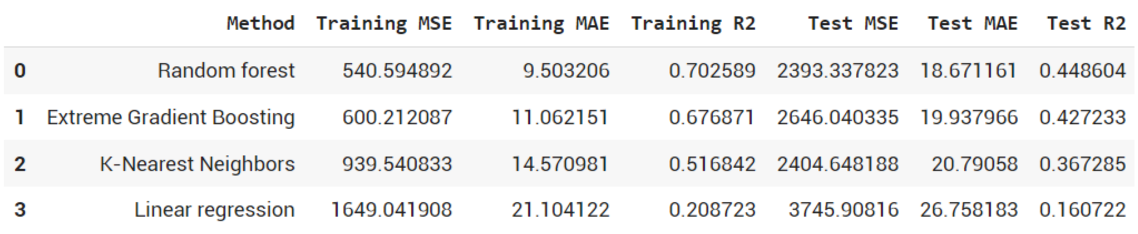
We can predict the deviations dx, dy, dz which are the difference between the intended FlightPlan point to the actual telemetry point at a given time, which in this case is the time stamp at the end of each segment.

Therefore, $dx = x2 - x1$, $dy = y2 - x1$, $dz = z2 - x1$. These deviations are the output labels that the model trains on. With an accurate prediction of dx, dy, dz, it is possible to obtain the telemetry point $x2, y2, z2$, $x2 = x1 + dx$, $y2 = y1 + dy$, $z2 = z1 + dy$.

Input data: *['JUNO', 'OMAHA', 'UTAH', 'SWORD', 'M300', 'M600', 'Mavic2', 'S9000', 'z0', 'x3', 'y3', 'z3', 'vel_fp', 'turn_fp', '3Ddist', '2Ddist', 't_seg_fp'],*

Output data: *[‘dx’, ‘dy’, ‘dz’]*

Below we can the performance of some of four models. These are models that accept multiple outputs as now we are predicting the three variables at the same time. We could predict one at a time but after having tested that, the results are worse. This is likely due to the relation between coordinates and the model benefits can establish links between the output variables and its magnitude.



	Method	Training MSE	Training MAE	Training R2	Test MSE	Test MAE	Test R2
0	Random forest	540.594892	9.503206	0.702589	2393.337823	18.671161	0.448604
1	Extreme Gradient Boosting	600.212087	11.062151	0.676871	2646.040335	19.937966	0.427233
2	K-Nearest Neighbors	939.540833	14.570981	0.516842	2404.648188	20.79058	0.367285
3	Linear regression	1649.041908	21.104122	0.208723	3745.90816	26.758183	0.160722

Figure 41. Performance of models for deviation prediction

As mentioned at the start of this position prediction section, the supposition about no strong relationship on between input and output data can be seen here below with the two best performing models. For instance, the XGB model gives almost the same importance to the feature *SWORD*, which is a categorical vertiport feature, as *x3* which describes the length in the X coordinate of the segment. This does not mean that the model misassigns the importance as it has performed decently, but it puts into perspective the lack of strong relationships.

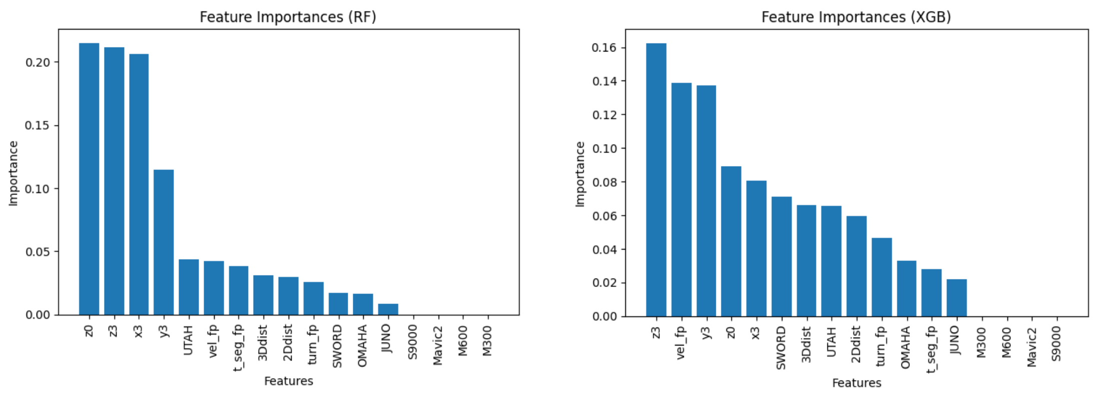
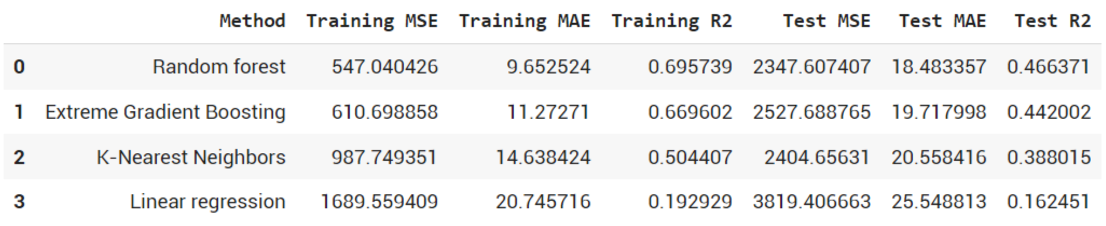


Figure 42. Feature importances for Random Forest and Extreme Gradient Boosting models.

We could try removing the less important features and adding other that might be relevant yet it is not simple with our data to obtain a feature that has a strong impact on the deviation of the drone. Maybe it could be due to wind conditions or the quality of the signal, among similar factors, that we do not have data on. By removing the drone models and adding some columns the results are similar. That is expected as these features do not really contribute to the final result.



	Method	Training MSE	Training MAE	Training R2	Test MSE	Test MAE	Test R2
0	Random forest	547.040426	9.652524	0.695739	2347.607407	18.483357	0.466371
1	Extreme Gradient Boosting	610.698858	11.27271	0.669602	2527.688765	19.717998	0.442002
2	K-Nearest Neighbors	987.749351	14.638424	0.504407	2404.65631	20.558416	0.388015
3	Linear regression	1689.559409	20.745716	0.192929	3819.406663	25.548813	0.162451

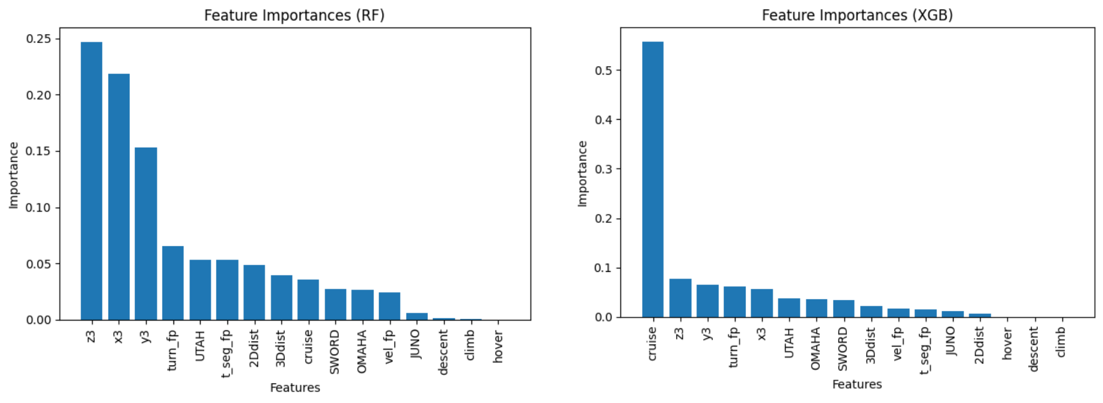


Figure 43. Slight variation in the results with different arrangement of data

It is interesting to note how the XGB model in this case makes the cruise feature as the most relevant. This can be associated to finding a clear relationship between deviations and cruise segments. It makes sense as deviations tend to be rather small and constant during cruise segments.

We can inspect the learning curves to see how the model is learning from this data. As seen in the figure below, it does seem like the models are slowly learning. There is a slow tendency to convergence yet the value is not impressively high. This suggest again that the model would highly benefit from more data to reach a higher scoring.

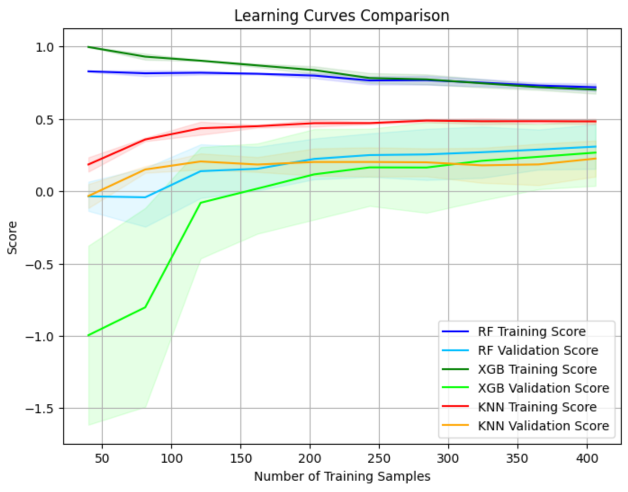


Figure 44

We have to be careful when choosing the features. If we include $x0, y0, z0$ and $x1, y1, z1$ to the input data, which might make sense at first as it represents the actual origin and end point of the segment, the performance will appear to clearly increase yet this is a clear case of overfitting the model.

	Method	Training MSE	Training MAE	Training R2	Test MSE	Test MAE	Test R2
0	Extreme Gradient Boosting	571.386192	10.863837	0.881327	2286.167121	19.423271	0.767509
1	Random forest	549.565675	10.017681	0.881263	2351.906279	21.162622	0.736521
2	Linear regression	1551.557913	21.31952	0.728789	3574.239761	27.335554	0.658907
3	K-Nearest Neighbors	1056.731453	16.947972	0.776253	2984.913061	26.393249	0.582479

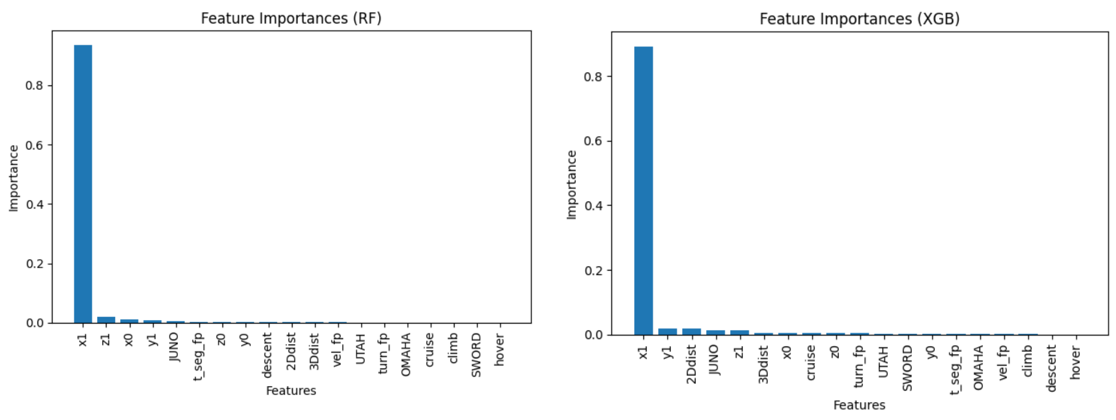


Figure 45. Models overfitting due feature memorization

We can clearly see that the model has increased its R2, almost doubling it, yet the errors on the test data remain the same or even higher. This is due to the memorization of a feature and predicting based on that feature. In this case, it is memorizing *x1* which provides no direct information to the deviation of the drone. The feature *x1* represents the X coordinate of end point of a segment and merely with a point in space it is not possible to make a prediction on the deviations. In the test data there are probably some points with very similar x1 as some of the trajectories share points and it makes the prediction entirely based on this feature. That explains why despite increasing the accuracy, the error is even higher. Unlike the case with t_fp, that closely related to the output variable t_tel when predicting time. In that case, the reason on why a feature stands out is feasible.

4.3. Airways safety margins

In order to establish a criterion when designing the spacing between areas, we can analyze the telemetry data against the FlightPlan to obtain valuable results that can be helpful in terms of safety and prevention of collisions. Also, this also ensures the telemetry will remain inside these margins. In a way this serves as an alternative to predicting positions.

If we can determine the area (or volume) where the drone will be flying with a high degree of confidence, we do not need to strictly know where it will be exactly located at a given time, instead we can guarantee that it will be inside a volume close to the trajectory. The volume can be a buffering of the segments by specifying distances or a tube-like shape with the segment in the center. In a way we can imagine trajectories as imaginary cylinders where the segment is the longitudinal axis. In this case, the imaginary cylinder is hollow and the drones would fly inside it.

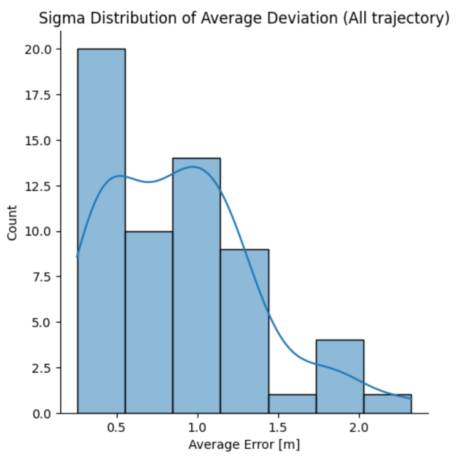


Figure 46. Sigma distribution of the deviation of the drones from the segments in meters.

This, in combination with the time predictions makes a pretty solid foundation for U-Space as we can give a prediction of the drones' real time in every waypoint and also specify a safety volume where the drone will be inside at that predicted time with confidence.

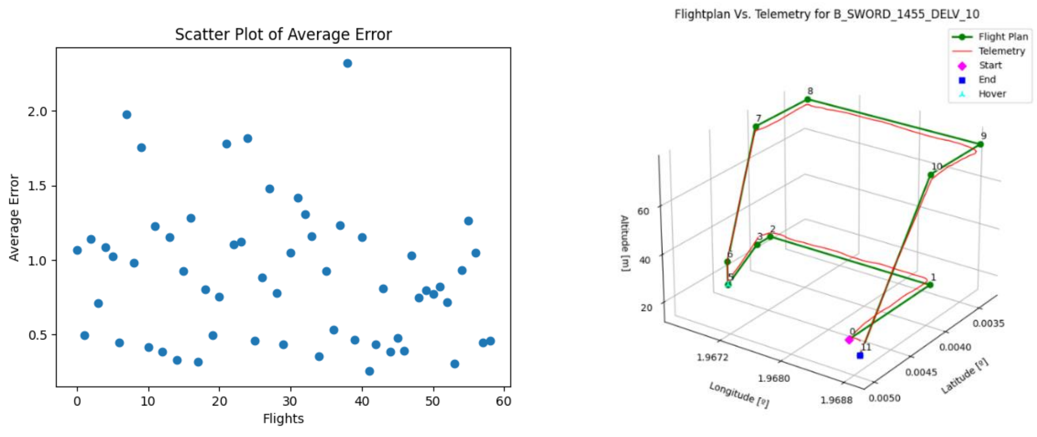


Figure 47. Average deviation [m] scatter plot and representation of a high average error flight

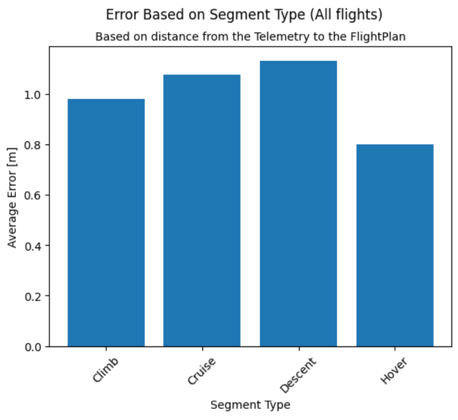


Figure 48. Average deviation from the telemetry to the flight plan segments for all flights

To better understand these results, it is helpful to plot the distribution by segment type. We have considered only for these plots the flights that have no contingencies or missing data. For reference, we are dealing with 354 cruise segments, 152 climb segments, 118 descent segments and 59 hoverings. The number of climbs and descents does not have to strictly match as in some case multiple climbs can be taken to reach a certain altitude, while the descent can be one in only one segment or vice versa.

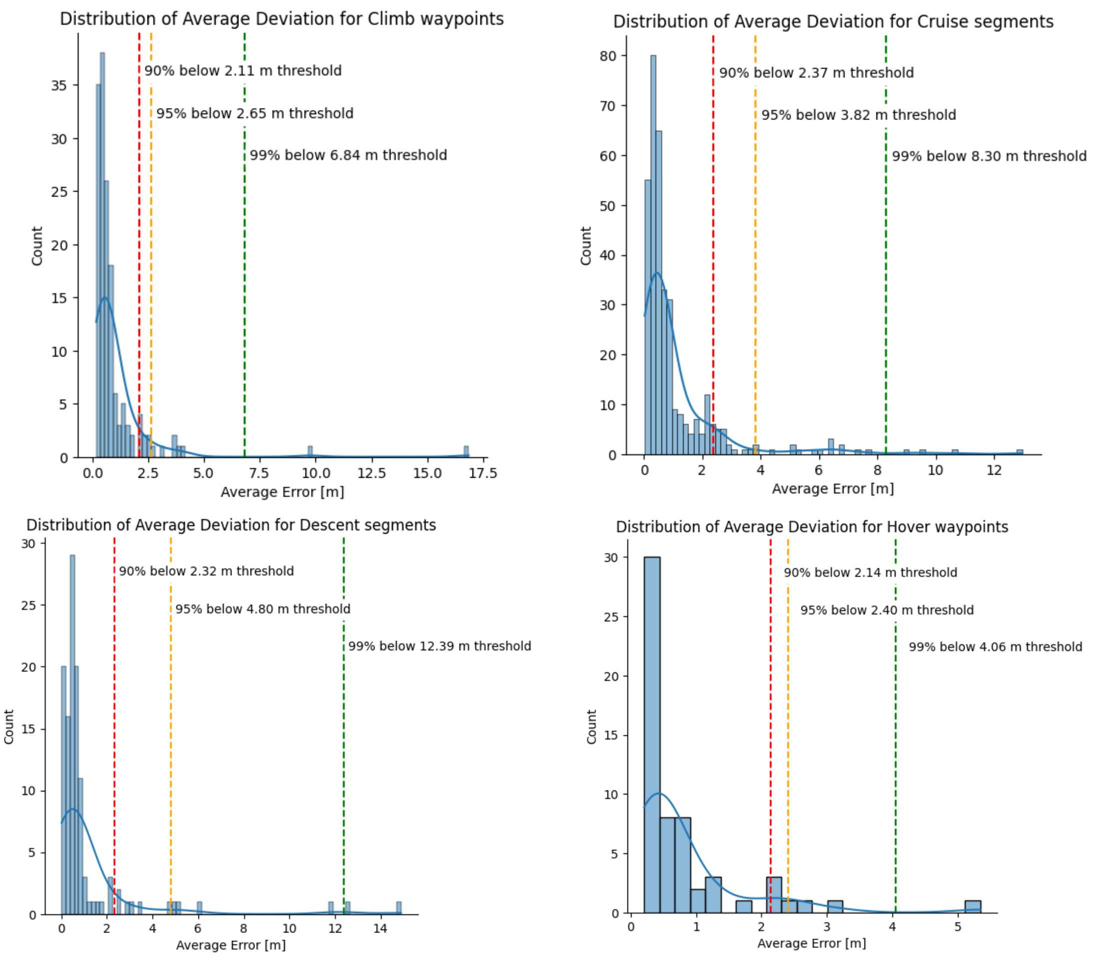


Figure 49. Distribution of average errors based on the type of segment with threshold for which average deviation is 90, 95 and 99 percent of the times.

CHAPTER 5. Conclusions and future work

Throughout the realization of this final degree project, we have managed to go from raw data in the form of flight plans and their associated telemetry, to a detailed analysis which is of high value to visualize the performance of the drones in terms of adherence to the flight plan and predicting times, which are essential for the development and improvement of the U-Space.

Several data visualization and observation functions have been made to clearly represent the data that we are dealing with. This has been indispensable to progress with the making of this project as the raw data itself is not intuitive at first sight and does require manipulation in order to get assumptions from it. In this kind of studies with experimental it is important to have a solid representation of the processing of the data as it is works well together with the numeric results that are obtained in each stage.

The time independent analysis done in Chapter 2 has set a base for us to build the rest of the project. To achieve objectives, observing the different relationships between the flight plan and telemetry is necessary. This analysis and developed algorithms make possible the relation between both kinds of data in the form of linking the indices, or points, of the telemetry to the flight plan despite the noise and smoothed shape that the real trajectory shows when the drone flies it. Without a clear correlation between this data, analyzing the data becomes an obstacle. The synchronization of the data also plays an important role on getting rid of the delay that the telemetry might experience, specially at the beginning of the trajectories and to train the models it is important to treat all telemetries the same, without any bias due to delay. And this does not only serve for the models but also for delimiting segments in telemetry and computing deviations which help determine the spacing of the airways that the drones use.

This project also contributes to the prediction of times for a given flight plan. Assessing the difference between the intended plan time and the actual telemetry is crucial for safety and capacity terms. With the use of various models, we have been able to obtain accurate time predictions based on the input of a flight plan after computing some features that can be obtained from it. The predictions have a remarkably good performance in comparison to the U-Space predictions, which satisfies the main goal of this final degree project. The high variation in the different telemetries of the flights makes it complex to always achieve low errors and there is room for improvement. Having more features in the raw could minimize these errors and increase accuracy. From the learning curves we can tell that also increasing the size of the dataset would be beneficial for the models to achieve convergence for the training data and validation data.

Related to what has been mentioned, when it comes to predicting positions becomes a complex task for supervised machine learning models. This is due to the fact that predicting the future position of telemetry for a whole trajectory based only on the intended flight plan is not an straightforward task as the deviations are highly irregular and variable during the trajectory. However, what we do to tackle this issue is to seek an alternative way of determining positions with some degree of uncertainty.

From the utilization of the data at out hand, we can define average deviations the drones experience from the intended segments of the flight plan. Plotting these deviations as distributions with threshold values ensures certain key parameters for the definition of the corridors composing the airways that can as well be interpreted as the spacing between airways, which can be beneficial for collision avoidance problems and maintaining safety. This is one of the fundamentals for the U-Space and being able to provide these insights is a valuable way of contributing to that.

Future projects can evolve from the basis of this project. Data analysis is time-consuming tasks and there are countless steps from the moment when the data is captured by the drone's sensor to having the data ready for a machine learning model. The developed algorithms are able to process the data after the initial stage of processing the KML files obtained from the readings of the drones. By having a set of data containing location coordinates and time, the algorithms and models used for this project can be used.

A way to improve this study is to gather more features that could be potentially useful for the machine learning models. These can range from meteorologic parameters that affect the adherence to the flight plan such as wind measurements, to specific parameters of the drone's battery level, quality of signal coverage, number of visible satellites for the positioning system of the drone, among others. Another beneficial improvement is to increase the size of the available dataset. The more data, as long as it is not repetitive, is a clear advantage for machine learning models as they are exposed to more combinations of the data and therefore learn to generalize for future unseen data.

In the area of prediction, future research can be done in the field of deep learning. A broad and powerful branch of machine learning falls on the algorithms of neural networks. Neural networks operate with a set of hidden neurons, or algorithms, layered together to create a complex network that acts as model that to make predictions. In addition to that, different kinds of neural networks can be used in order to learn from a time series and predict the future values. A way of implementing these kinds of neural networks is to make predictions based on previous data. In that case, previous information of the real time telemetry can be provided as an input to the model, alongside other features, to output the continuation of the telemetry. Precisely, LSTM (Long Short-Term Memory) and RNN (Recurrent Neural Networks) can be used to achieve said goal, [14], [15]. A network able to determine future positions can be used in combination with the developed machine learning models to have a more robust behavior.

REFERENCES

[1] EUROCONTROL, “Challenges of Growth 2013 - Task 4: European Air Traffic in 2035,” 2013.

[2] SESAR, Joint Undertaking, “SMART ATM U-space”, (2023). https://www.sesarju.eu/U-space

[3] Thipphavong, D.P., Apaza, R.D., Barmore, B.E., Battiste, V., Belcastro, C.M., Burian, B.K., Dao, Q.V., Feary, M., Go, S., Goodrich, K.H., Homola, J.R., Idris, H.R., Kopardekar, P., Lachter, J., Neogi, N.A., Ng, H.K., Oseguera-Lohr, R.M., Patterson, M.D., & Verma, S.A. (2018). Urban Air Mobility Airspace Integration Concepts and Considerations. 2018 Aviation Technology, Integration, and Operations Conference.

[4] pandas Development Team. "pandas: Data Analysis Library." 2023. Web. Retrieved from https://pandas.pydata.org/.

[5] "Geographic coordinate conversion." Wikipedia: The Free Encyclopedia. Wikimedia Foundation, [2023]. https://en.wikipedia.org/wiki/Geographic_coordinate_conversion#From_geodetic_to_ECEF_coordinates.

[6] "World Geodetic System" Wikipedia: The Free Encyclopedia. Wikimedia Foundation, [2023]. https://en.wikipedia.org/wiki/World_Geodetic_System#WGS84

[7] "Local tangent plane coordinates" Wikipedia: The Free Encyclopedia. Wikimedia Foundation, [2023]. https://en.wikipedia.org/wiki/Local_tangent_plane_coordinates

[8] Paglione, Mike & Oaks, Robert. (2007). Implementation and Metrics for a Trajectory Prediction Validation Methodology. 10.2514/6.2007-6517.

[9] B. Johnen and B. Kuhlenkoetter, "A Dynamic Time Warping algorithm for industrial robot motion analysis," 2016 Annual Conference on Information Science and Systems (CISS), Princeton, NJ, USA, 2016, pp. 18-23, doi: 10.1109/CISS.2016.7460470.

[10] Romain Tavenard, . "An introduction to Dynamic Time Warping." . (2021). Available at: https://rtavenar.github.io/blog/dtw.html

[11] B. Johnen and B. Kuhlenkoetter, "A Dynamic Time Warping algorithm for industrial robot motion analysis," 2016 Annual Conference on Information Science and Systems (CISS), Princeton, NJ, USA, 2016, pp. 18-23, doi: 10.1109/CISS.2016.7460470.

[12] Al-Naymat, Ghazi & Chawla, Sanjay & Taheri, Javid. (2012). SparseDTW: A Novel Approach to Speed up Dynamic Time Warping. Conferences in Research and Practice in Information Technology Series. 101.

[13] Georgia Tech Professional Education. (2023). Introduction to Machine Learning and Three Common Algorithms.

[14] Dai, Wei & Zhang, Mingcheng & Low, Kin. (2022). Data-Efficient Modeling for Precise Power Consumption Estimation of Quadrotor Operations Using Ensemble Learning. 10.48550/arXiv:2205.10997.

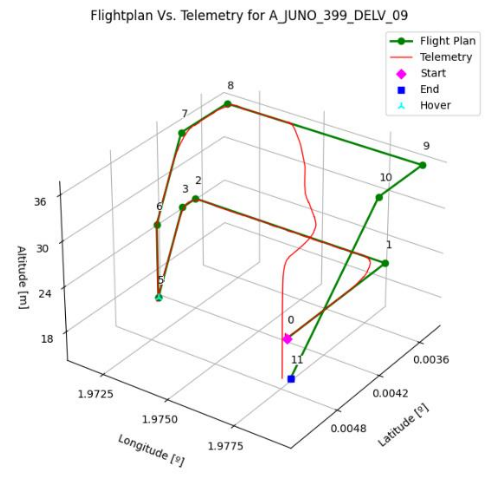
[15] Anush Manukyan, Olivares-Mendez Miguel, Holger Voos, Matthieu Geist. Real time degradation identification of UAV using machine learning techniques. International Conference on Unmanned Aircraft Systems (ICUAS), 2017, Miami, United States. ffhal-01629680f

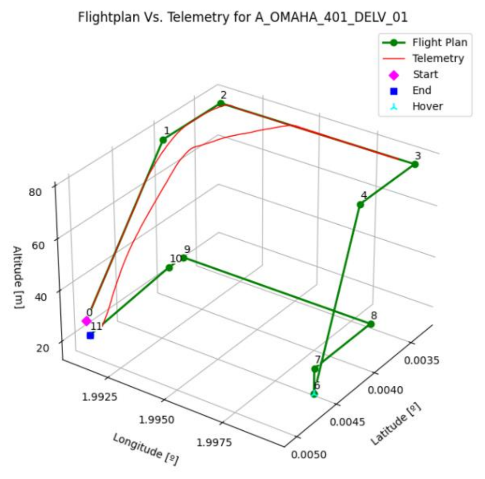
[16] H. -C. Choi, C. Deng and I. Hwang, "Hybrid Machine Learning and Estimation-Based Flight Trajectory Prediction in Terminal Airspace," in IEEE Access, vol. 9, pp. 151186-151197, 2021, doi: 10.1109/ACCESS.2021.3126117.

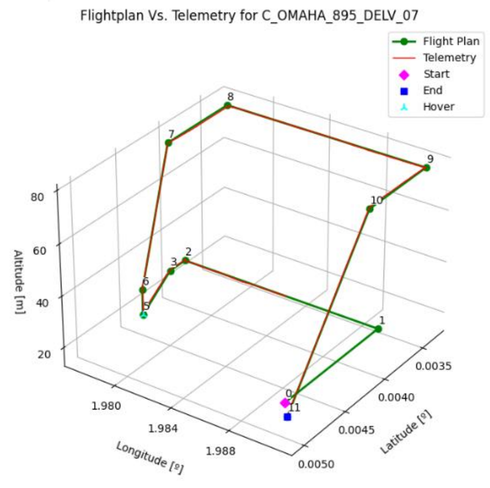
[17] Shi, Zhiyuan & Xu, Min & Pan, Quan & Yan, Bing & Zhang, Haimin. (2018). LSTM-based Flight Trajectory Prediction. 1-8. 10.1109/IJCNN.2018.8489734.

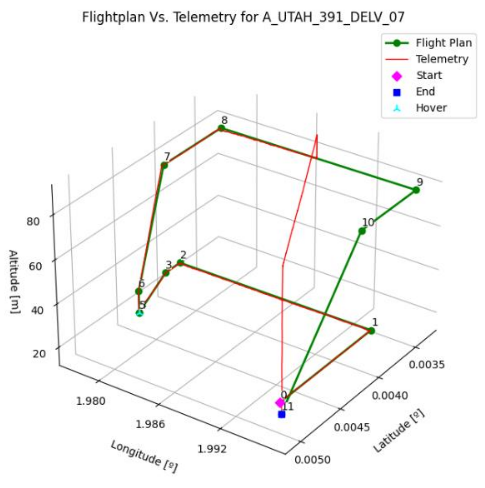
ANNEXES

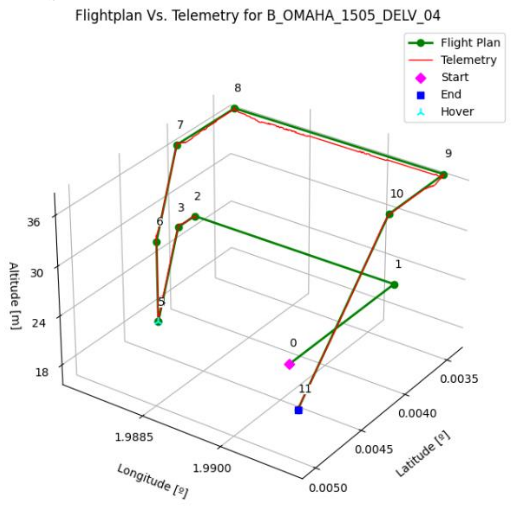
Contingency flights (including the ones with missing data)

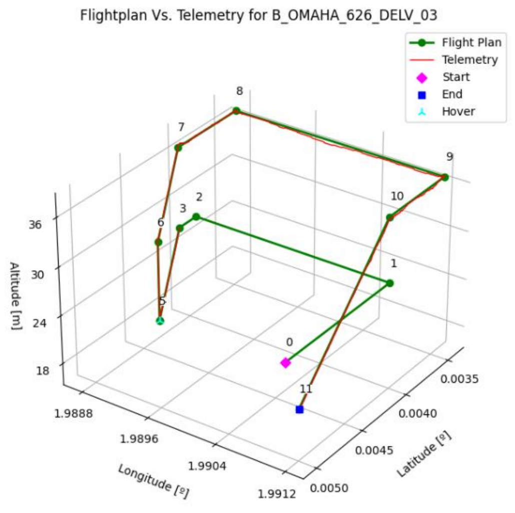




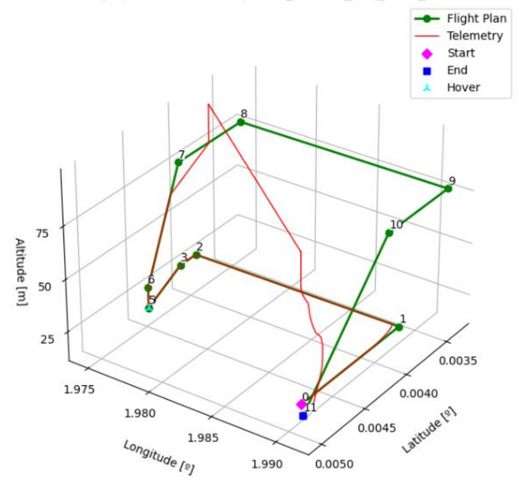




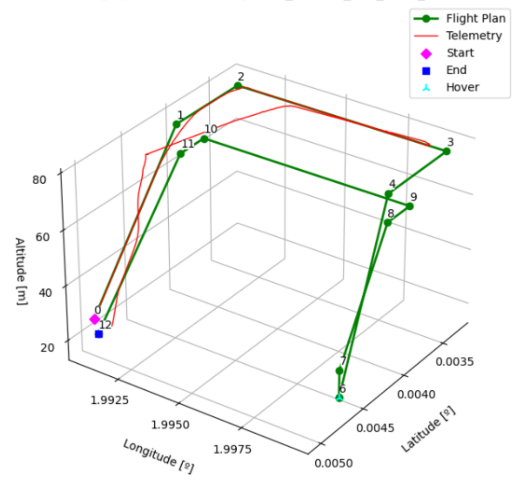




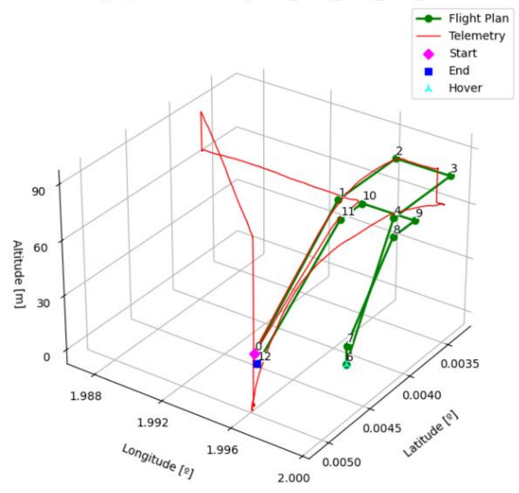
Flightplan Vs. Telemetry for A_OMAHA_411_DELV_08



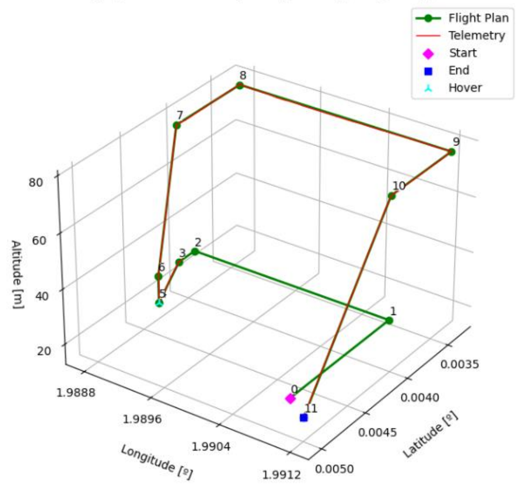
Flightplan Vs. Telemetry for A_OMAHA_128_DELV_01



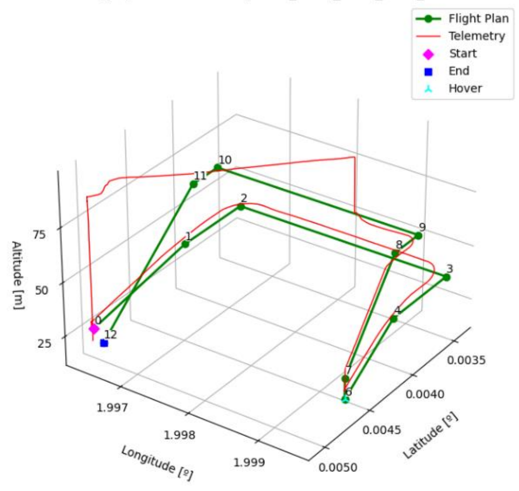
Flightplan Vs. Telemetry for B_UTAH_1006_DELV_01



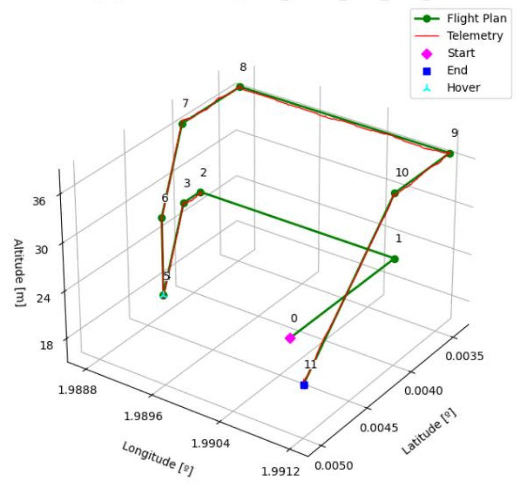
Flightplan Vs. Telemetry for B_OMAHA_1556_DELV_03



Flightplan Vs. Telemetry for B_UTAH_1485_DELV_01



Flightplan Vs. Telemetry for B_OMAHA_1002_DELV_03


Flight Plan
Telemetry
Start
End
Hover
Altitude [m]
Longitude [º]
Latitude [º]