Aerodynamic Model Identification of DarkO: A Mini VTOL

Master's Degree Thesis

Author: FAVALLI Federico (Student ID: 250890)
Supervisor: Ph. D. BRONZ Murat
Co-Supervisor: Prof. GUGLIERI Giorgio

Politecnico di Torino (DIMEAS)
École Nationale de l'Aviation Civile

Faculty of Aerospace Engineering

A.Y. 2018 - 2019

Acknowledgements

I thank those who know they must be thanked

Abstract

Small VTOL airplane models are starting to be very widespread as they lend very well for different purposes. Unfortunately, however, the aerodynamics of very small aircraft is much more complicated to understand than popular large aircraft, as the parameters involved are very different in nature. For this reason an experimental investigation of their aerodynamic behaviour is required in order to fully understand their behaviour.

Contents

List of Figures

[1]By Tosaka - Opera propria, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=4454089

[2]By Johannes Thinesen - The SFF photo archive., CC BY-SA 2.5, https://commons.wikimedia.org/w/index.php?curid=1615013

[3]By NASA - http://lisar.larc.nasa.gov/UTILS/info.cgi?id=EL-2001-00399, Public domain, https://commons.wikimedia.org/w/index.php?curid=12486606

[4]Designed by Gannett Fleming, https://www.uber.com/us/en/elevate/uberair/

[5]https://lilium.com/the-jet

[6]Krossblade Aerospace

[7]http://www.aeroskytech.com/english/firstnotions/anglesen.png

[8]By Motion Imagery Standards Board (MISB) - Extracted from File:MISB Standard 0601.pdfOriginally from http://www.gwg.nga.mil/misb//docs/standards/ST0601.8.pdf, Public Domain, https://commons.wikimedia.org/w/index.php?curid=39884471

List of Tables

1 Introduction

1.1 Definition of VTOL and description

A VTOL [10] vehicle is an aircraft capable of hover and perform vertical take-off and landing. This means that to take off and land it does not need the relative runways. Some of them, such as some helicopters, can only operate in the VTOL configuration as they have no trolley and, consequently, are unable to operate on the ground. Others, on the other hand, equipped with a trolley can also operate in the CTOL (conventional), STOL (short), STOVL (short takeoff and vertical landing) configuration.

Within the global VTOL category, there are also several variants of this config-

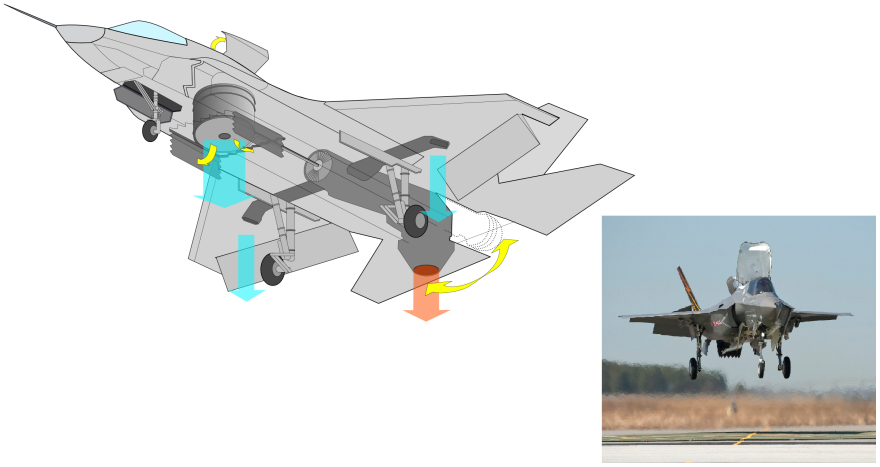


Figure 1: Lockheed Martin F-35 Lightning II in vertical takeoff[9]

uration. They can be fixed-wing aircraft, helicopters and other types of powered rotors aircraft. Lighter-than-air aircraft may also belong to it, as they can hover, take off and land in approach/departure configuration. However, the VTOL category is a sub-category of the broader V/STOL (vertical and/or short takeoff and landing).

The VTOL family can be divided into two large subgroups: the rotorcraft and the powered-lift.

The rotorcraft are a family of heavier-than-air aircraft that use lift generation through wings, which in this case are rotating wings, to operate in flight. More specifically, following the official definition of the ICAO (International Civil Aviation Organisation) a rotrcaft is defined as "supported in flight by the reactions of the air on one or more rotors".

Belonging to the rotorcraft group are:

- *helicopter*: an aircraft whose engine(s) causes the rotor to rotate throughout

Figure 2: Fairey Rotodyne - Example of a VTOL rotorcraft[10]

the flight. This allows the helicopter to perform landing and vertical take-off, but also to hover and move forward, back and sideways. This allows these aircraft to reach and operate in places where fixed-wing aircraft cannot arrive;

- *autogyro*: it is characterised by a rotor that moves due to the aerodynamic forces acting on it and, therefore, without the aid of a propulsion unit. Propulsive unit which is instead necessary for thrust generation. What differentiates it from a helicopter is that it needs an air flow on the rotor to allow it to rotate. It is therefore not really a VTOL, as to perform a vertical take-off it needs an external aid for the rotation of the rotor and for the next vertical landing it needs a precise control of the rotation and the pitch;

- *gyrodyne*: it is a VTOL very similar to a helicopter, since the main rotor is set in rotation by a dedicated motor, in which in addition there are one or more thrusters that generate the forward thrust required for movement during the cruise flight. The lift, on the other hand, is generated by the union of that coming from the rotor and the conventional wings with which it is equipped;

- *rotor kite*: it consists of a version of the autogyro in which, however, no propulsion unit appears. In fact, while in the autogyro the thrust is generated by a motor, in the case of the rotor kite it moves either following a launch from another aircraft or the action of the natural wind which then rotates the rotor naturally.

The power-lifts are those VTOLs that differ from the rotorcraft in that they work differently during horizontal flight. In fact, in this phase of flight they depend mainly on a generation of lift by motors and not by rotating wings.
Belonging to this family we can find:

Figure 3: Ling-Temco-Vought XC-142A - Example of a VTOL powered-lift aircraft[11]

- *convertiplane*: it takes off and lands like a helicopter, or under the action of the lift generated by rotors, but for horizontal flight it turns into a plane with fixed-wing lift. In this configuration the tiltrotor and the tiltwing the rotor changes orientation, moving forward to act as a propeller. In other design, instead, it is possible to have a ducted fan in which the propeller is surrounded by a circular duct to reduce losses at the ends;

- *tail-sitter*: it "sits" on the tail pointing towards the other for take-off and landing. The entire aircraft is then rotated to allow horizontal flight;

- *vectored-thrust*: this technique is used for both aircraft engines and rocket engines. It is based on varying the direction of the outgoing jet according to need. As regards VTOL it can be varied from horizontal to vertical;

- *lift jet*: it is a light engine used to generate the thrust necessary for VTOL operations. The peculiarity is that it is then switched off when switching

to conventional horizontal flight. It can also be used as an auxiliary unit for the vectored-thrust technique;

- *lift fan*: it is an airplane configuration in which the lifting facelifts are positioned in an otherwise conventional wing or inside the fuselage. In this configuration, the aircraft takes off using the lift generated by this fan and then transitions to a fixed wing configuration.

1.2 State of the art

Due to their versatility and their better adaptation to different environments compared to CTOL aircraft, VTOLs [3] are paying great attention to the way they operate transport in the future, even in the near future. Many companies, among which we can find for example Uber Elevate (service under development of the well-known Uber transport company, increasingly expanding), Lilium (German company with the objective of producing Lilium Jet, VTOL electric), Kitty Hawk (company US manufacturer of electric aircraft, including the VTOL model of the Kitty Hawk Cora) and Volocopter (a German company that promotes its own Volocopter 2X, an electric helicopter derived from its previous version), announced that they have projects or have already tested or produced VTOL in some of the different facets. Even the most famous and traditional companies in the aerospace sector have moved in this direction, such as Airbus and NASA, which are in fact experimenting with this technology. However, the objectives are different. Some companies, in fact, aim to perform medium-long range flights remaining within a metropolitan area. Taking up the companies mentioned above, instead we see how, for example, Lilium has as its objective flights that cover much greater ranges, even aiming at the London-Paris route. Uber, on the other hand, has announced that it wants to start using flying taxis by 2020, whereas Kitty Hawk has for now "only" obtained permission to test autonomous flight aircraft in New Zealand. Furthermore, many of these companies collaborate with leading companies in the sector, such as the well-known Bell Helicopters or Embraer.

However, the debate persists on the subject of autonomous flight or not. Many

Figure 4: Design of a Uber Skyport[12]

companies conceive the flight completely automated in the future, while others,

instead, believe that the pilot is necessary inside the aircraft or, at least, in a control centre until the automated flight will be more advanced and complete. However, some companies, including Lilium, are designing their aircraft to make piloting easier, which therefore requires a simple Sport Pilot License. In this regard, for example, US regulations for VTOL prototypes are not very clear at the moment. It seems clear, therefore, that there is a need for more intervention by the competent authorities in order to develop new rules as this trend towards automated flight is increasingly growing.

Instead, what is known is the tendency to go towards the electric. In fact, the

Figure 5: Lilium Jet in VTOL configuration[13]

VTOLs currently in the design and development phase are mostly equipped with electric motors and batteries, thus opposing the traditional liquid fuel, which is therefore going to disappear. The current problem, however, is that of battery storage. To overcome this drawback, the aircraft are designed using light materials.

1.3 Advantages and disadvantages

VTOL [1] is challenging. Technically speaking, it is much more difficult than the conventional flight (winged flight), but on the other side it has several advantages:

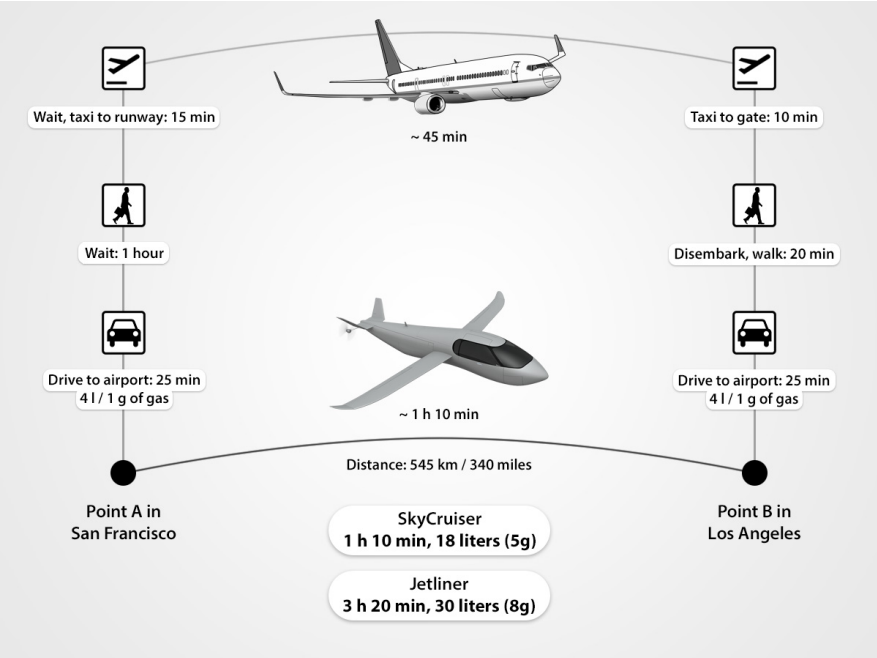


Figure 6: Direct point to point travel with SkyCruiser vs indirect travel via airport[14]

- *time and fuel economy*: VTOL allow an aircraft to takeoff almost from every location, certainly more than a CTOL which instead require an airport. This means that passengers can travel from point A to point B directly;
- *efficiency and speed*: VTOL aircraft can be designed in such a way that its wings are designed to optimise efficiency and speed, since they do not influence the takeoff and landing. Moreover, if the remaining components do not cause much additional drag, then it is possible to achieve a real net drag reduction, making the aircraft more efficient and, if necessary, faster;
- *safety*: a large part of all air accidents occur during their conventional take-off and landing. Since the aircraft is accelerated and decelerated while high in the air, making these phases vertical can significantly reduce this risk, adding margin for mistakes and recovery.

Other minor advantages [3], linked above all to the mode of use of such aircraft, may be the shorter time taken for the journey, particularly if one is in

a metropolitan context, the reduction in greenhouse gas emissions since most VTOLs are designed to work in electric mode and, finally, noise reduction compared to traditional helicopters.
On the other hand, however, the disadvantages associated with these aircraft are also present and are not negligible. In particular we have:

- limited flight time and range, as they are linked to battery storage and weight problems;

- even if they do not need airports in which to take-off and landing manoeuvres, it will be necessary to construct or delimit specific landing areas, thus requiring definitions and controls in this sense from the authorities;

- the pilots are not completely eliminated, or at least they are not in the near future, until the automated flight is complete;

- the regulations for aerospace control must therefore be negotiated and reviewed;

- any price that is too high to use this service, inevitably linked to many other factors, could limit the demand;

- the climatic conditions could affect the safety and, therefore, the reliability of this service, thus limiting the demand again;

- last but not to be underestimated, it must reach the consensus of the community, a factor that could vary greatly from one region to another.

1.4 Description and purpose of the research

1.4.1 Motivation

Small VTOL airplane models are starting to be very widespread as they lend very well for different purposes. Unfortunately, however, the aerodynamics of very small aircraft is much more complicated to understand than popular large aircraft, as the parameters involved are very different in nature. For this reason an experimental investigation of their aerodynamic behaviour is required in order to fully understand their behaviour.
Several studies on small planes have been done over the years. A study [8] was done using a model of 37.5 cm of wingspan and 11.5 g of weight available on the market. To capture the flight data, they used a Vicon [9] infrared camera system that tracked reflective markers on the aircraft. In this way the entire flight was recorded. This allowed them to analyse the aerodynamic behaviour of the aircraft and in particular the moment of reaching the stall.
Another study [6] relatively related to the behaviour of VTOLs was conducted by Manchester et al. Inside a wind tunnel and using an adjustable test stand at different angles of attack, they analysed an aircraft model with a forward-sweep wing variable between 0° and 25° and, comparing it with conventional aircraft, they have noted that this configuration ensures an increase in lift in the post-stall area, or rather that which characterises the perching manoeuvres, very useful for landings in areas that are difficult to reach in other ways.
A very interesting study [4], on the other hand, sought to highlight the differences and limitations encountered in trying to study the aerodynamics of an aircraft through experimental methods. In particular, in this research they compared the results from a wind tunnel test and those obtained, instead, by a numerical simulation. It highlighted the fact that, however accurate, a numerical simulation remains an approximation of reality and, as a result, admits limits that should not be underestimated. In particular, they showed that the drag was under-predicted and the stability coefficients required further validations.
The complexity of the VTOL aircraft and what has been analysed and presented in the previous studies leads us to understand how, in order to achieve sufficient reliability of the results, we must analyse them in a real case scenario, trying not to rely too much on approximate models. Furthermore, as regards the use of a wind tunnel, it is limiting for two main reasons. The first is that not everyone can have access to a wind tunnel of sufficient size to be able to simulate a free flow comparable with a real flow. As a result, many rely on a small wind tunnel. This implies, however, the establishment of wall effects that only create noise in the

data detected by the tests. The second is that the sensors used to measure forces and moments acting on the aircraft, for example, are very expensive in economic terms and, therefore, not available to everyone.

1.4.2 Objectives

In order to solve the problems presented in 1.4.2 related to the study of this type of aircraft, in this research we used a motion capture system of 16 cameras built by OptiTrack [7]. The system could track the aircraft during the whole indoor flight, recording its positions and quaternions. These data and their derivatives have been then compared to the ones recorded by the GPS and IMU mounted on board the aircraft in the same indoor flight, so as to be able to understand if the latter is a reliable model and therefore be able to use it in the future to study the performance of the aircraft using only data coming from outdoor flights, hence without the help of an external instrument, which allow a more complete study of the phenomena that affect the vehicle.

2 Experimental setup

2.1 Description of the experimental environment

Throughout the course of the research, two main working environments were used: the "Volière" of the ENAC of Toulouse and the aerodrome "Club Eole de Muret".
As for the Volière, it consists of a building equipped and furnished in order to contain within it a dedicated, delimited and protected area used for the flight of drones or any other model of aircraft that can perform a flight in a limited space.
As shown in figure 7, the area dedicated to the flight is rectangular in shape and

Figure 7: Top view of the Volière

delimited by protective nets. The metal structure surrounding the flight zone serves primarily as a mounting support for the infrared cameras of the OptiTrack motion capture system. The laboratory is equipped with a total of 16 cameras (visible as blue lights in the figure), four for each side of the structure, all pointing inwards. Moreover, on the structure, a normal camera is also mounted which allows the resumption of the flights performed inside the area.
Inside the building it is also possible to find a control room (figure 8), that is to say an area dedicated to series of computers that allow the user to interact with the OptiTrack system. In fact, during the flight, 9 markers are positioned on the aircraft used (figure 9 and 10) in order to allow OptiTrack to trace, at a frequency of 1000 Hz, the aircraft recognising the virtual reproduction dictated by these markers. On the computers inside the office, it is possible to monitor the flight of the aircraft (visible precisely by means of markers) and act on certain parameters of the aircraft. For example, if the aircraft is flying in autopilot mode, the user can correct the speed, the altitude, etc. Or, he can change the display,

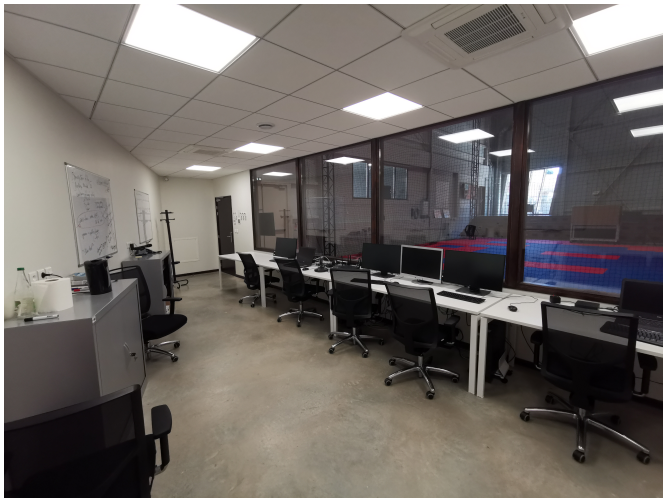


Figure 8: Inside view of the office in the Volière

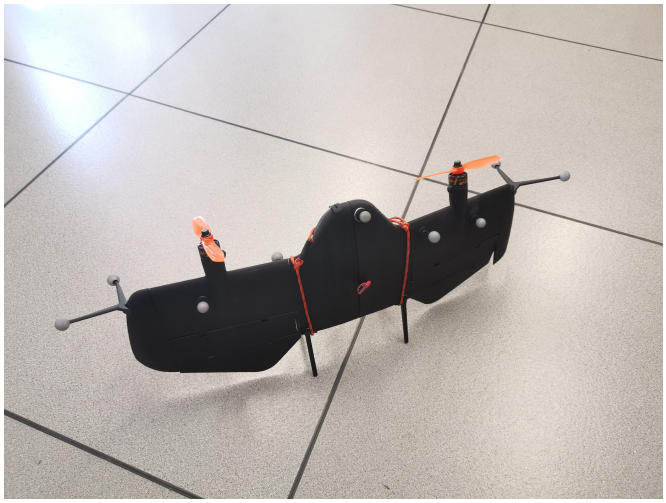


Figure 9: DarkO with the OptiTrack markers

Figure 10: Detail of DarkO markers

decide whether to focus on a particular marker, etc.

In the control room, connected to these computers there is also the red button

Figure 11: Detail view of the button in the office of the Volière

(figure 11) through which OptiTrack data recording is started and terminated manually, an action that, as it will be explained later in this document, will lead to some mismatches between these recordings and those made through the GPS and IMU mounted on board the aircraft during the flight.

As for the second working environment, it simply consists of an aerodrome dedicated to model aircraft and drone flight tests. . It is located a few minutes from

Figure 12: Aerodrome "Club Eole de Muret"

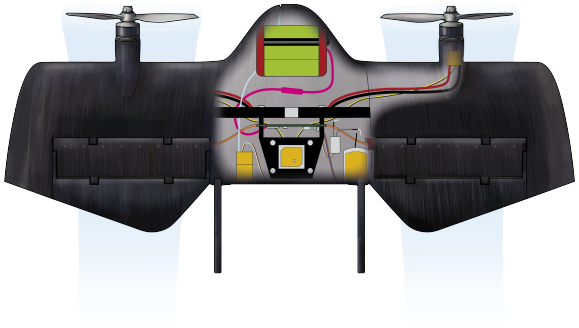
Toulouse and has a main runway (figure 12) through which the aircraft can take off and land safely. For the DarkO flight this track is not necessary, as it takes off and lands vertically from the pilot's hand since it is not equipped with supports or landing gears, but this environment is necessary to be able to fly especially safely.

3 DarkO Hybrid MAV [5]

This section introduces the DarkO hybrid MAV and all its physical and geometric parameters. We briefly present its manufacturing process and the characteristics of motors, propellers, servos, and battery that are used during the experimental flight tests. Additionally, we describe the control surface design of the DarkO that includes a double flap deflection.

3.1 Setup and specifications

Throughout the whole study, we have used the DarkO vehicle which is a tail-sitter configuration consisting of two motors, positioned in front of the wing, and two exceptionally large double-flapped control surfaces. Mission definition of DarkO has been mainly oriented for forward flight with the capability of taking off and landing vertically. The frame completely manufactured by 3-D printing method using Onyx material. Figure 13 shows the printed pieces that is assembled in order to build the whole frame. The shell structure for the wing and the fuselage halves are manufactured as $0.7mm$ thick skins, and the spar is manufactured with the addition of unidirectional concentric carbon fibres embedded into Onyx material. This method ensures to have a sufficiently rigid airframe that supports aerodynamic forces and yet also flexible enough to absorb harsh impacts during landing and test flights.



Mass: 0.492 Kg
Wingspan: 0.55 m
Wing Area: 0.0743 m^2
Mean Chord: 0.13 m
Propellers: 2-blades Bullnose 5x4.5
Motors: T-Motor Brushless F30 2800KV
Servos: MKS DS65K 0.2s/60°
Battery: 3 cells 12V 3500 mAh

Figure 13: General DarkO hybrid MAV's specifications.

The different physical and geometric parameters of the DarkO MAV, are described in the Table 1. Inertia coefficients were estimated by using the classical pendulum method and the aerodynamic coefficients calculated from the open-source program XFOIL. These different parameters were used in the simplified hybrid MAV model, described in the following section, in order to develop, anal-

yse and validate the proposed control architecture, as realistic as possible, in simulation before the experimental flights.

Table 1: DarkO MAV parameters used during flight simulations

Parameters	**Values**	**SI Units**
Mass (m)	0.492	[Kg]
Mean Chord (c)	0.135	[m]
Wingspan (b)	0.55	[m]
Wing Area (S)	0.0743	[m^2]
J_{xx}	0.00493	[Kg m^2]
J_{yy}	0.00532	[Kg m^2]
J_{zz}	0.00862	[Kg m^2]
J_p	5.1116e-06	[Kg m^2]
k_f	5.13e-6	[Kg m]
k_m	2.64e-7	[Kg m^2]
C_{d0}	0.133	No units
C_{y0}	0.145	No units
C_l	[0.47; 0.00; 0.00]	No units
C_m	[0.00; 0.54; 0.00]	No units
C_n	[0.00; 0.00; 0.52]	No units

3.2 Control surface design

A particular feature that is required by the tail-sitter configuration is to generate excessive amount of pitching moment in order to transition mainly from forward flight phase to hovering flight phase. Therefore, DarkO frame's control surfaces have been designed as double-flap which has a passive mechanical constant ratio. Traditionally, multi-section flaps have been designed for lift enhancement, however in our case the design objective is to generate as much positive pitching moment as possible without having a massive flow separation on the bottom surface of the airfoil.

3.3 Actuators and attitude dynamics

The attitude dynamic is controlled according to the actuation principle described in the Fig. 15, 16, and 17. Hybrid MAVs are characterised as nonlinear systems with high coupled dynamics. In fact, pitch and roll angles are controlled respectively by symmetric and asymmetric flap deflections who are dependants of the

Figure 14: Double-flapped control surface

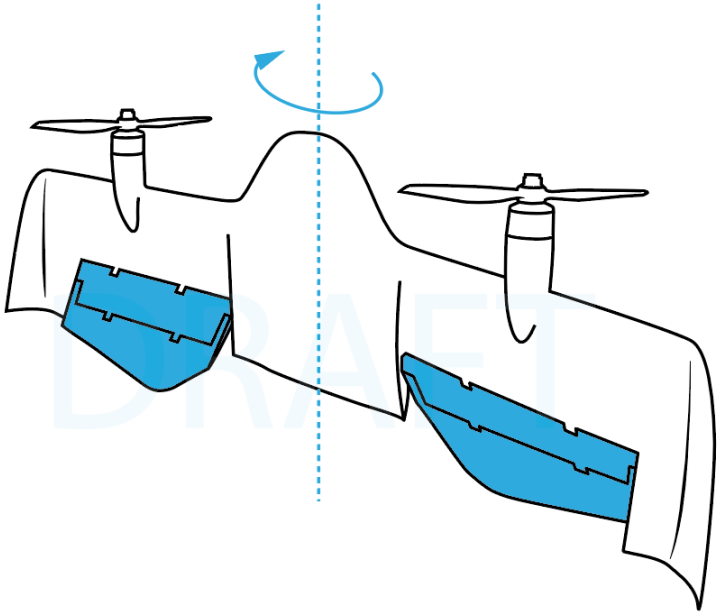


Figure 15: Roll angle dynamic

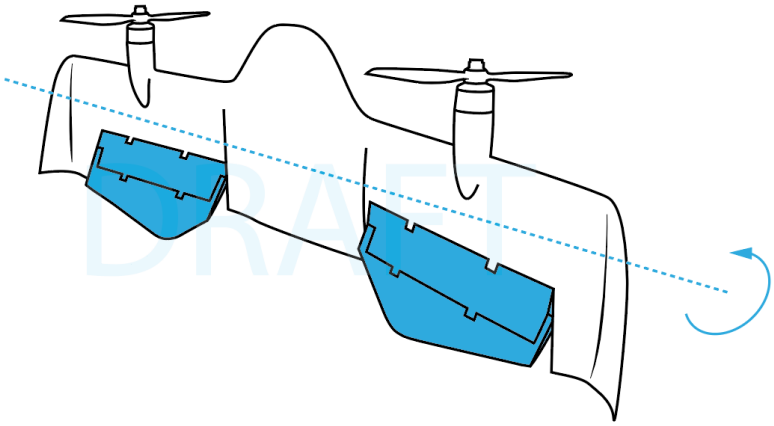


Figure 16: Pitch angle dynamic

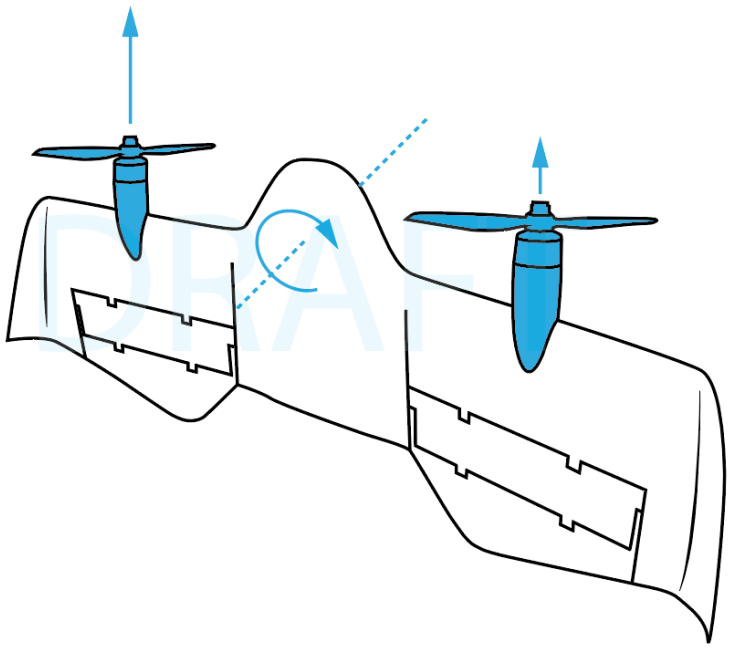


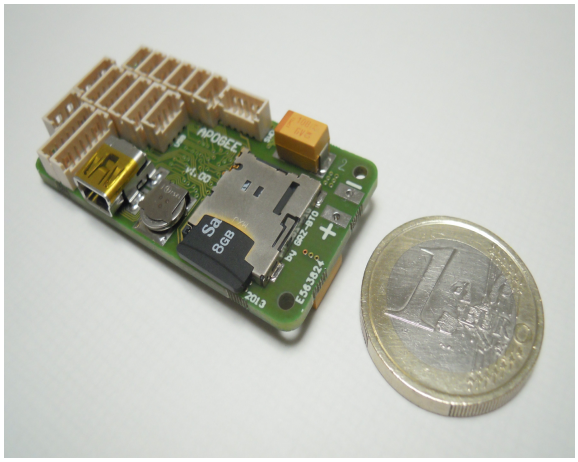
Figure 17: Yaw angle dynamic

propeller slipstream. The differential thrust in order to control the yaw angle modifies the propeller slipstream impacting the control-effectiveness of the flaps as well as the dynamic of both pitch and roll angles.

3.4 On-board avionics

The DarkO MAV is equipped with an *Apogee v1.00* board, presented in the Fig. 18, that contains a Cortex M4 168 MHZ processor to run the *Paparazzi* open-source autopilot system, which includes algorithms for state estimation, control laws, servo and motor drivers, software for communication, etc. In addition, the *Apogee v1.00* board is equipped with a SD logger which allows us to record the flight data for flight post-processing analysis.

Inertial measurement units (IMUs) typically contain rate-gyroscopes and accelerometers on three axes, measuring angular velocities and linear accelerations respectively. By processing signals from these devices, with attitude and heading reference system (AHRS) and inertial navigation system (INS), it is possible to obtain the attitude orientations, velocities and positions of an air vehicle. The main features of each sensor device embedded in the *Apogee v1.00* board, is presented in the Table 2.



STM32F405RGT6 Cortex M4 168MHz processor
9(6) DOF integrated IMU MPU-9150(6050)
1 x Barometer/altimeter MPL3115A2
1 x MicroSD card slot
4 bit SDIO interface (high speed data logging)
6 x Servo PWM outputs
3 x UART, 2 x I2C bus, 1 x SPI bus
10.4 grams
53 mm x 25 mm

Figure 18: Overview of *Apogee v1.00* autopilot from *Paparazzi Autopilot system*

Table 2: *Apogee V1.00* embedded sensors

Sensors	**Device**	**Noise**	**Bias**
Accelerometer	MPU-9150	400 $(\mu g/\sqrt{Hz})$	150 (mg)
Rate-Gyro	MPU-9150	0.005 $(°/s/\sqrt{Hz})$	20 $(°/s)$
Magnetometer	MPU-9150	N/A	N/A
GNSS position	NEO-6M	$\sigma = 2.5$ (m)	0 (m)
GNSS velocity	NEO-6M	$\sigma = 0.1$ (m/s)	0 (m/s)

4 Performance of activities

4.1 Indoor flight analysis - OptiTrack and IMU data comparison

Identifying the aerodynamic characteristics and performance of a VTOL (vertical takeoff and landing) plane is very complicated as it is not enough to rely only on indoor flights, as atmospheric factors that can disrupt the flight cannot be taken in account. On the other hand, it is not even possible to rely only on outdoor flights and therefore on a IMU data detection system as this is not always reliable. The objective of this part of the research is, therefore, to verify and validate the measurements obtained from the IMU using a sophisticated OptiTrack high-precision motion camera system as a comparison and to understand, therefore, the usability in future of only outdoor flights.

4.1.1 Data processing

In order to obtain greater ease of use, the script used for the calculations has been divided into two macro-parts that can be consulted in appendices A and B, dividing in this way the linear and the angular quantities codes. Nevertheless, from this moment on, reference will be made to the study as if it has been performed in the order of explanation so as to make the discussion more fluid and logical.

4.1.1.1 IMU - SD card

Having obtained the files containing the data of interest for the study, these must be manipulated in order to make them usable for the calculations.
The first step is to take the LOG file from the SD card mounted on the aircraft. Being unusable in its original form, it is necessary to open it and save its contents in two different steps, so as to be able to view in the first place the integer data and, in the second, the non-integer data (float). Subsequently, the data obtained are resized and placed in specific arrays in order to make the following recall and use easier and more intuitive. In particular the fundamental data coming from this file are: the operating frequency of the IMU (equal to 500 Hz), the quaternions and the linear accelerations. Furthermore, these data are recorded according to the NED reference system (North, East, Down), i.e. the one that will be used throughout the course of the study, therefore they do not need further

modification.
Having the need subsequently to calculate the angular velocities that characterise the rotations of the aircraft, the quaternions recoverable in the data file of the SD card are converted into Euler angles, using the eq. 1, found in the literature.

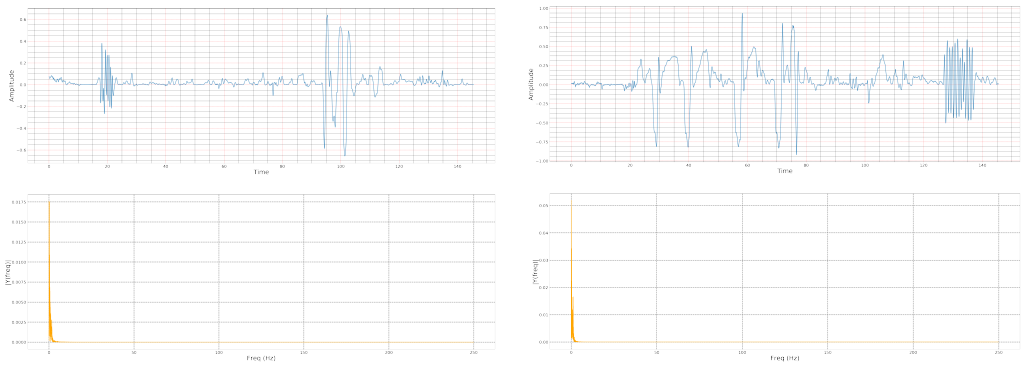
$$\begin{Bmatrix} \phi \\ \theta \\ \psi \end{Bmatrix} = \begin{Bmatrix} atan2[2(q_0q_1 + q_2q_3), 1 - 2(q_1^2 + q_2^2)] \\ asin[2(q_0q_2 - q_3q_1)] \\ atan2[2(q_0q_3 + q_1q_2), 1 - 2(q_2^2 + q_3^2)] \end{Bmatrix} \quad (1)$$

The unusual use of the atan2 function in place of the more usual atan can be noted. This is due to the fact that by using atan2 it is possible to generate all the possible orientations without the limitations of the range $[-\pi/2, \pi/2]$, limitations presented instead by the atan function.
At this point, the data (Euler angles, speeds and accelerations) need to be filtered as they are characterised by a non-indifferent noise level and that makes it difficult to use and, above all, to read and compare them. To do this, 1 € Filter [2] is used, a code developed specifically for the Python platform that can reduce the noise present in a series of data. Before applying this code to the data, however, it is necessary to conduct an FFT (Fast Fourier Transform) in order to identify the minimum cutoff frequency for each set of data taken into consideration, so as not to risk excluding sensitive values from the analysis. The graphs and results returned by the FFT code are presented in figures 19 anr 20.

As visible, the code returns a graph of the trend of the amplitude of the array and its frequency spectrum. From this last graph it is possible to identify a range in which the aforementioned minimum cutoff frequency is found. Once this range has been identified, it is necessary to go back to the 1 € Filter configuration and try manually with different frequencies in order to find the most correct one for each data series. The different values of the minimum cutoff frequencies found are the following:

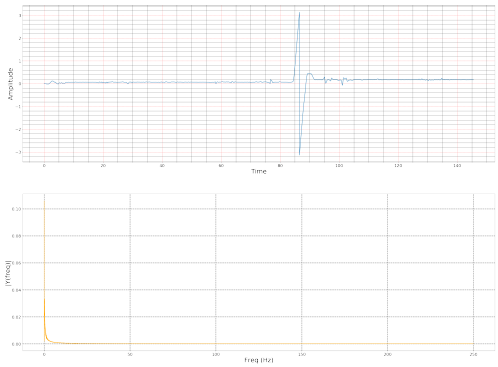
Table 3: Cutoff frequencies for SD data

Parameter	**Cutoff Frequency [Hz]**
Roll angle	1
Pitch angle	0.3
Yaw angle	0.3
Ax	0.1
Ay	0.1
Az	0.1



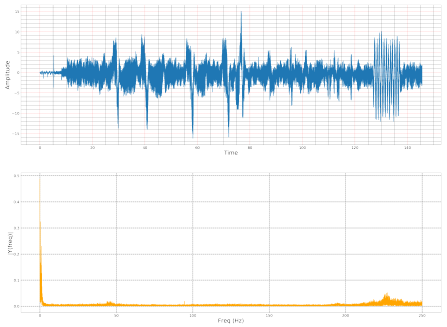
(a) *IMU Roll angle*

(b) *IMU Pitch angle*

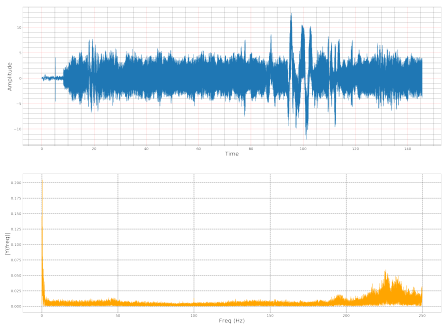


(c) *IMU Yaw angle*

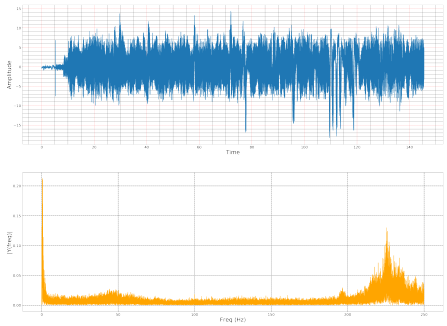
Figure 19: Euler angles FFT output for the IMU



(a) *FFT output for the SD X axis linear acceleration*



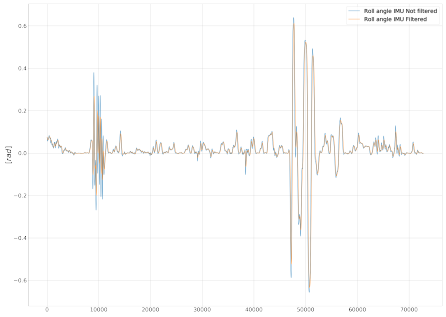
(b) *FFT output for the SD Y axis linear acceleration*



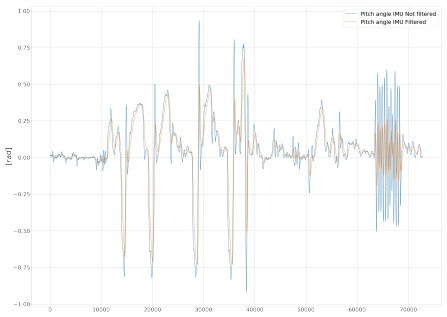
(c) *FFT output for the SD Z axis linear acceleration*

Figure 20: Accelerations FFT output for the IMU

In figures from 21 and 22 it is therefore possible to see the comparison of the data before and after having been filtered.



(a) *Roll angle not filtered vs Roll angle filtered for the SD card*



(b) *Pitch angle not filtered vs Pitch angle filtered for the SD card*

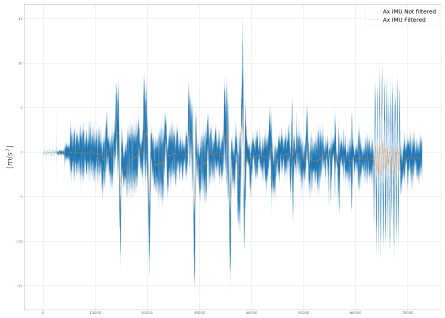
(c) *Yaw angle not filtered vs Yaw angle filtered for the SD card*

Figure 21: Euler angles filtered vs not filtered for the IMU

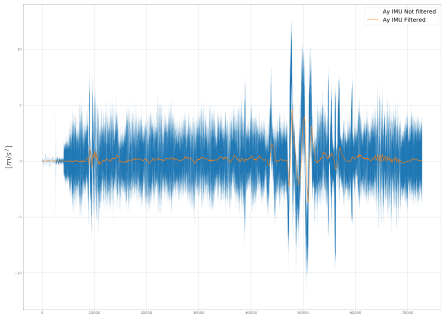
The last step is to calculate the angular velocities and accelerations, starting from the Euler angles, as these will then be used later to calculate the moments of the aircraft. To do this, first of all Pandas is used, a python library that allows the user to work with DataFrames instead of simple arrays. In this way it is possible to use the appropriate function provided by the Pandas library which allows to derive a series of data with respect to a time range, which in this case is the frequency with which the data is recorded. Again, the example for the roll first and second derivatives from the code is presented below.

$$sf = 500$$
$$diff\_period = sf$$
$$p\_sd = roll.diff(periods{=}diff\_period)$$
$$pdot\_sd = p\_sd.diff(periods{=}diff\_period)$$

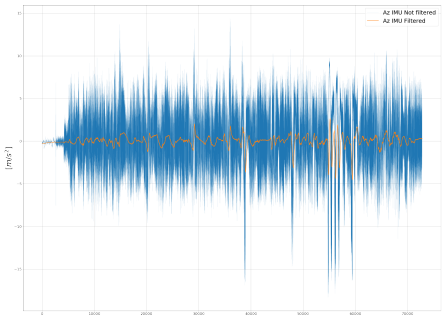
where sf, equal to the differentiation period, is the sample frequency of the IMU, and p_sd and pdot_sd respectively are the angulare speed and acceleration.



(a) *Ax not filtered vs Ax filtered for the SD card*



(b) *Ay not filtered vs Ay filtered for the SD card*



(c) *Az not filtered vs Az filtered for the SD card*

Figure 22: Accelerations filterd vs not filtered for the IMU

In this way, the angular velocities are obtained with a first derivation and the angular accelerations with a second derivation. These data must not be filtered as they come from the Euler angles previously filtered.

4.1.1.2 OptiTrack

Now, the same procedure is repeated with the data file obtained from the OptiTrack camera system. This time, this file is in CSV format and, therefore, directly readable by the code. It appears as follows: it is characterised by seven columns in total which, in order, present the four parameters of the quaternions (X, Y, Z, W) and the three coordinates representing the position of the aircraft (PX, PY, PZ). In this case, the data was not recorded directly in NED, but in a variant of this reference system. Therefore, before proceeding with any other step, it is necessary to adjust this component orientation. The transformation, for quaternions and position coordinates, is as follows:

$$q_0 = q_0$$
$$q_1 = -q_3$$
$$q_2 = q_1$$

$$q_3 = -q_2$$
$$Px = -PZ$$
$$Py = PX$$
$$Pz = -PY$$

Moreover, another problem arises. In fact, during the execution of the flight, there has been small moments in which OptiTrack has lost the connection with the aircraft, thus producing small gaps in the file. These gaps could be dangerous in the subsequent computations of the data, especially in the removal of the noise as it would stop when it meets the gap. It is therefore necessary to fill these gaps manually within the file. However, these are intervals that do not exceed ten data lines and can therefore be corrected quickly.
Anyway, in this case, the file is directly read using the Pandas library, so as to obtain an overall DataFrame of the file right from the beginning. Since the data coming from OptiTrack can be considered exact, it is possible to use the positions (PX, PY, PZ) obtained from the system to plot what was the flight conducted by DarkO. The result is visible in figure 23 as a 3D view (coloured so as to be able to distinguish the curve without showing a uniform spot) and in figure 24 where this flight has been broken down into the three basic planes (XY, YZ, XZ), so as to better understand the manoeuvres made by the aircraft.

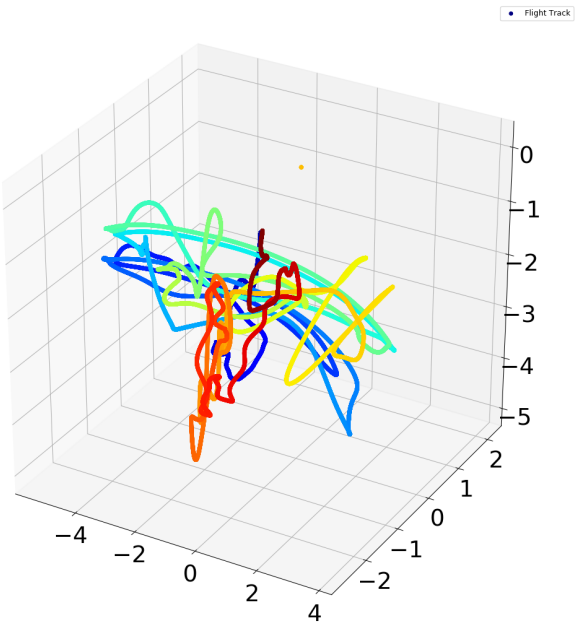
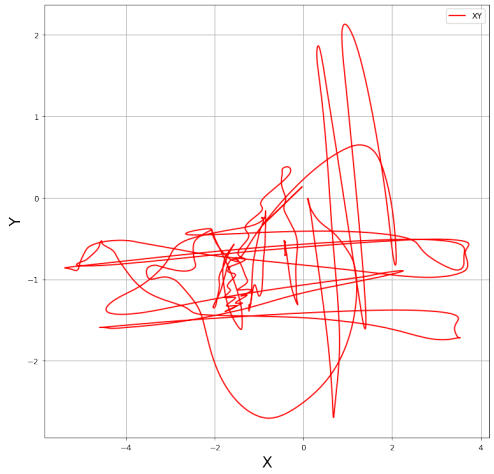
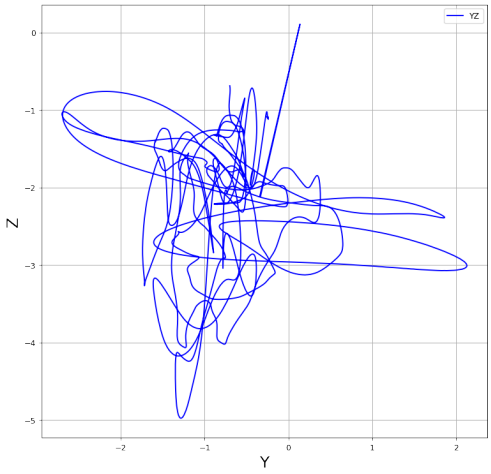


Figure 23: Flight track recorded by OptiTrack

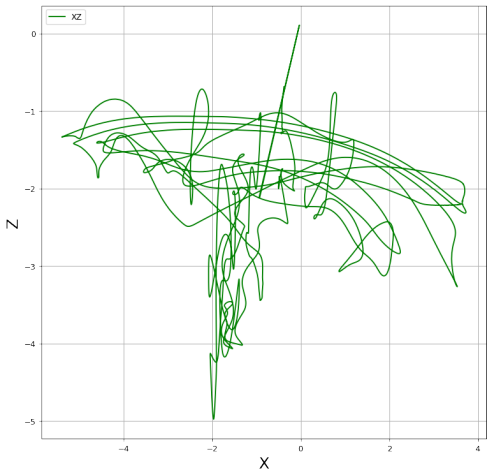
At this point, as done for the data coming from the SD card, the quaternions are used to return to the formulation with the Euler angles. The expression used is the same as in the equation 1.



(a) *XY plane of the flight track recorded by OptiTrack*



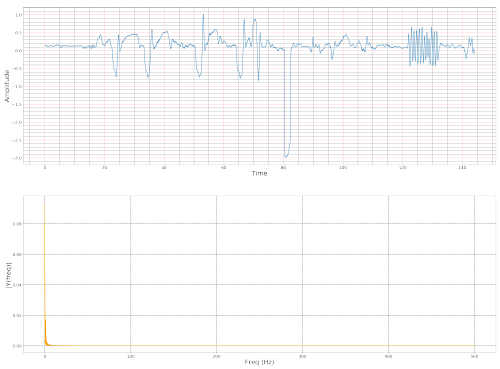
(b) *YZ plane of the flight track recorded by OptiTrack*



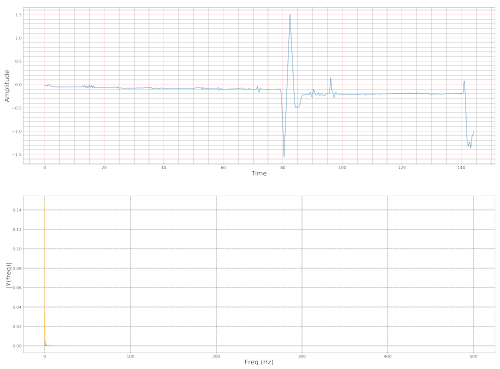
(c) *XZ plane of the flight track recorded by OptiTrack*

Figure 24: 2D visualisations of the flight track

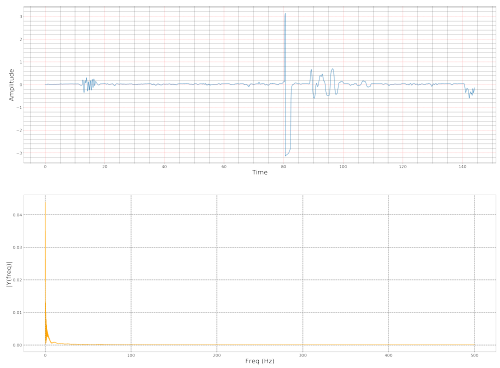
Also in this case a reduction of the noise present in the data series is necessary. However, unlike the first case, for the data coming from OptiTrack it will be applied only to the Euler angles coming from the quaternions, since the positions have a clean and smooth trend, optimal for their following use. Therefore, the FFT is again conducted to find the values of the range in which it is possible to identify the minimum cutoff frequencies. The FFT outputs can be displayed in figure 25.



(a) *FFT output for the OptiTrack roll angle*



(b) *FFT output for the OptiTrack pitch angle*



(c) *FFT output for the OptiTrack yaw angle*

Figure 25: Euler angles FFT output for the IMU

Once these ranges have been found, manual trials are conducted again to identify the optimal value of this frequency, values that are shown in the following table.

Table 4: Cutoff frequencies for OptiTrack data

Parameter	**Cutoff Frequency [Hz]**
Roll angle	0.3
Pitch angle	0.3
Yaw angle	0.4

In figure 26 it is therefore possible to see the comparison of such data before and after having been filtered using again the 1€ Filter code.

(a) *Roll angle not filtered vs Roll angle filtered for OptiTrack*

(b) *Pitch angle not filtered vs Pitch angle filtered for OptiTrack*

(c) *Yaw angle not filtered vs Yaw angle filtered ffor OptiTrack*

Figure 26: Euler angles filtered vs not filteres for OptiTrack

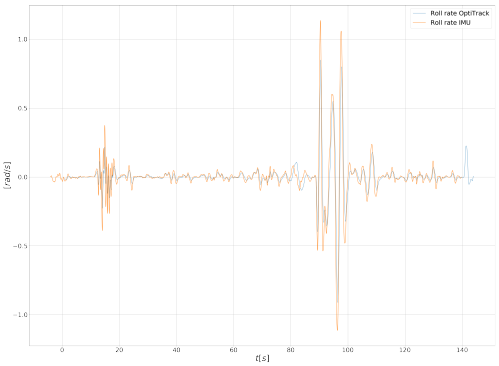
It is necessary, now, to derive the DataFrames related to these angles in order to obtain their rates with respect to the time. Angular velocities and accelerations therefore emerge from the derivations. The procedure is equal to the one shown above, with the difference that in this case the reference time will not be the same since the OptiTrack data are recorded with a frequency of 1000 Hz. In this case, however, in addition to deriving the Euler angles, it is also necessary to derive the positions of the aircraft in order to obtain the linear velocities and accelerations, as they are not provided by the OptiTrack system. Each DataFrame corresponding to a position is derived a first time to get the speeds and a second time to get the accelerations, as shown in the following expressions, which are again a small representation of what is present in the code (Appendix A).

$$sf = 1000$$
$$diff\_period = sf$$
$$VX = PX.diff(periods{=}diff\_period)$$

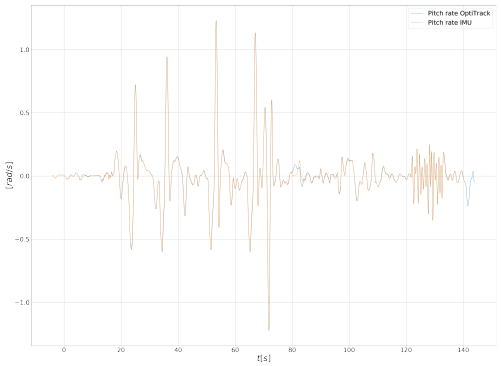
$$VY = PY.diff(periods{=}diff\_period)$$
$$VZ = PZ.diff(periods{=}diff\_period)$$
$$AX = VX.diff(periods{=}diff\_period)$$
$$AY = VY.diff(periods{=}diff\_period)$$
$$AZ = VZ.diff(periods{=}diff\_period)$$

4.1.2 Velocities and accelerations comparison

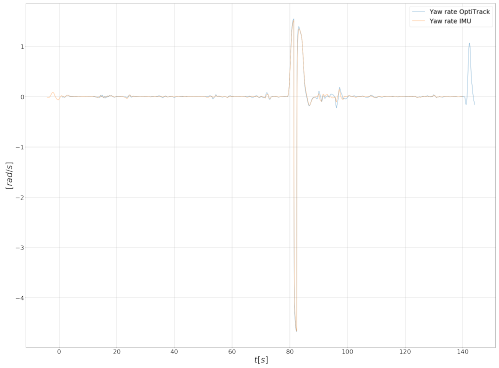
Having now obtained all the data necessary for the subsequent calculation of aerodynamic forces and moments, it is good to proceed first by comparing these results. In particular, it is necessary to see if the quantities obtained are comparable or not and, therefore, to have an idea in principle of what can be expected from them. In fact, depending directly on them, if the speeds and accelerations were totally different, the forces and moments would in turn be completely different and, consequently, such a study would be a failure. The angular velocities (Euler angles rates) are then analysed. They can be displayed in figure 27.



(a) *Roll rate comparison between OptiTrack and IMU*



(b) *Pitch rate comparison between OptiTrack and IMU*



(c) *Yaw rate comparison between OptiTrack and IMU*

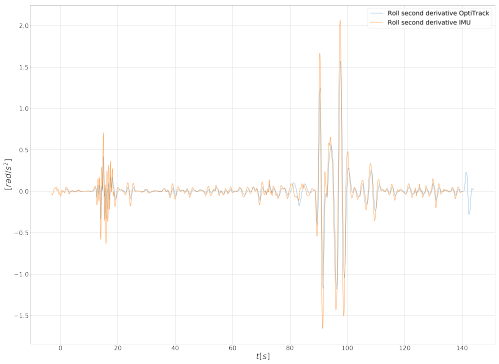
Figure 27: Angular velocities comparison between IMU and OptiTrack

The first thing that catches the eye is the slight horizontal offset between the

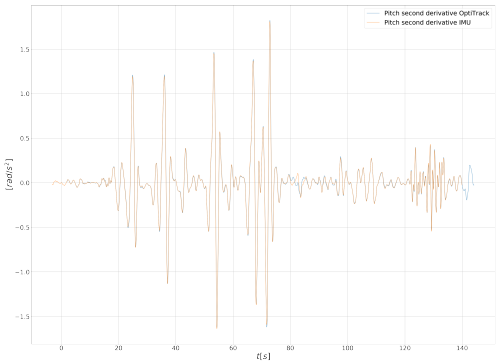
curve produced by OptiTrack and the one produced by the IMU. In fact we can see how the latter starts before the former. In fact they have a relative offset between them of about 5.1 seconds, translated in 5.1 horizontal coordinates. This is not a problem and it is not abnormal, as the IMU recording started when the aircraft started flying, while OptiTrack registration was started manually. Since the second operation is inevitably subject to human error, it is normal that the two recordings did not start at the same time. However, as already mentioned, this does not represent in any way a constraint for the purpose of the study and through the part of the code related to the graphics it can easily be eliminated. The second feature that catches the attention is the similar and in some places practically identical course of the two curves, but also the small differences in other sections. As for the differences they are mainly due to one principal factor, that is the vibrations of the aircraft occurred during the flight. In fact, being mounted on board, the IMU is subject to all the phenomena that affect the VTOL model and is therefore affected in the recording of flight data. These phenomena, on the other hand, are almost imperceptible at a distance and are therefore not displayed by OptiTrack. Furthermore, differences that can be noticed in the peaks (maximum and minimum) may have been created during the filtering of data from noise. As a result, they could be eliminated by conducting a more accurate and less approximate filtering.

As for the second derivatives of the Euler angles, or angular accelerations, it can be seen in figure 28 that they have the same characteristics as the first derivatives. In fact, the curves show the same trend and have some differences in the peaks. In particular, a greater difference can be noticed in the first section of the two curves relating to the roll angle rate. In fact, the peaks of this section of the two curves are characterised by very different amplitudes. This is a clear example of how data filtering has affected their next comparison. If for example a lower cutoff frequency lower for IMU data had been assumed, lower intensity peaks would have been obtained for the IMU curve. However, the remaining trend would have been affected and the comparison would not have had the validity that it now presents.

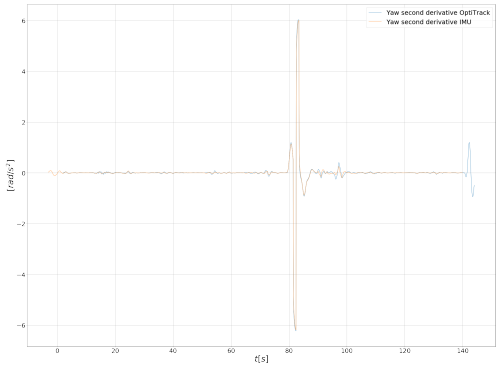
As far as linear accelerations are concerned, they show different comparison characteristics with respect to the angular quantities previously discussed. In fact, as can be seen in the figure 29, the two curves, the one relative to OptiTrack and the one relative to the IMU, globally have the same identical trend. But, analysing instead in detail, the values point by point are different, a very accentuated characteristic for example in the accelerations of the X axis. A feature



(a) *Roll second derivative comparison between OptiTrack and IMU*

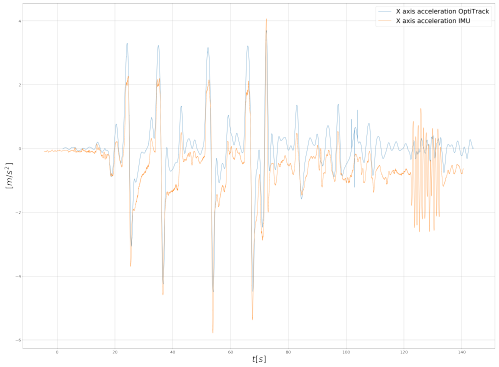


(b) *Pitch second derivative comparison between OptiTrack and IMU*

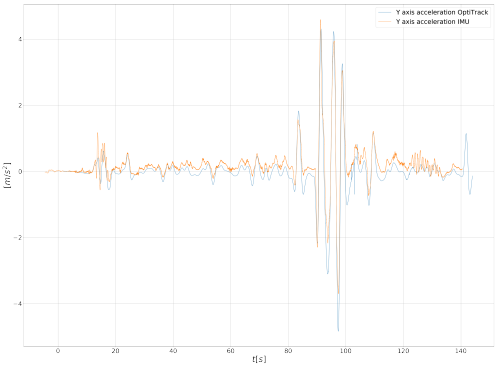


(c) *Yaw second derivative comparison between OptiTrack and IMU*

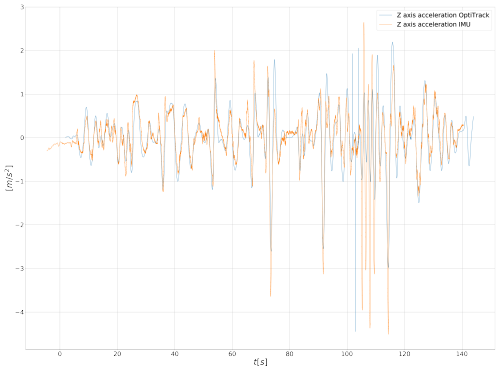
Figure 28: Angular accelerations comparison between IMU and OptiTrack



(a) *X acceleration comparison between OptiTrack and IMU*



(b) *Y acceleration comparison between OptiTrack and IMU*



(c) *Z acceleration comparison between OptiTrack and IMU*

Figure 29: Linear accelerations comparison between IMU and OptiTrack

which, on the other hand, is reduced to the minimum in accelerations along the Z axis, even if in the peaks of the zones where there are sudden changes of sign, a not inconsiderable difference can be noticed, feature due again mainly to data filtering. At any rate, although characterised by these easily visible differences, the latter are much smaller in magnitude than the width of the value of the variable in the point. Consequently they are easily negligible for the purpose of the study and it can be said that these accelerations are comparable.

4.1.3 Results and discussion

At this point, since these are the only variables present in the formulations of the forces and the aerodynamic moments that will be calculated later, since the other quantities involved are constant (mass, density, surface, etc.), the comparison phase can be considered concluded. What can be concluded at the end of this chapter is that the quantities taken into consideration (speeds and accelerations) are totally comparable. In fact, as mentioned, they always have the same pattern and in most cases (e.g. the angular velocities and accelerations) have even superimposed curves between OptiTrack and IMU. This was not obvious at the beginning of the study, in fact the factors that can alter the measurements of the IMU are numerous, for example the simple vibrations given by the propellers of the aircraft. Instead, it has proved to be very reliable and exact, a factor which therefore leads to assume that it can be used in outdoor flights relying solely on the data recorded by this IMU. Analysing the following formulas,

$$F = [a_x \ a_y \ a_z] \ m \tag{2a}$$

$$M_p = M_x = \dot{p} I_{xx} + qr(I_{zz} - I_{yy}) \tag{2b}$$

$$M_q = M_y = \dot{q} I_{yy} + pr(I_{xx} - I_{zz}) \tag{2c}$$

$$M_r = M_z = \dot{r} I_{zz} + pq(I_{yy} - I_{xx}) \tag{2d}$$

it can be seen how the calculated quantities influence the aerodynamic forces and moments. Starting from the first (2a), it is easy to understand how these forces, deriving from the accelerations of the two systems multiplied by the same mass of the aircraft (constant quantity and less than the unit), once plotted they present even smaller differences than those seen in the case of accelerations, since precisely equal to the latter reduced in size by the mass. As for the aerodynamic moments, the same explanation can be applied. In fact, the angular velocities and accelerations present in the expressions presented (2b - 2d) are multiplied by the

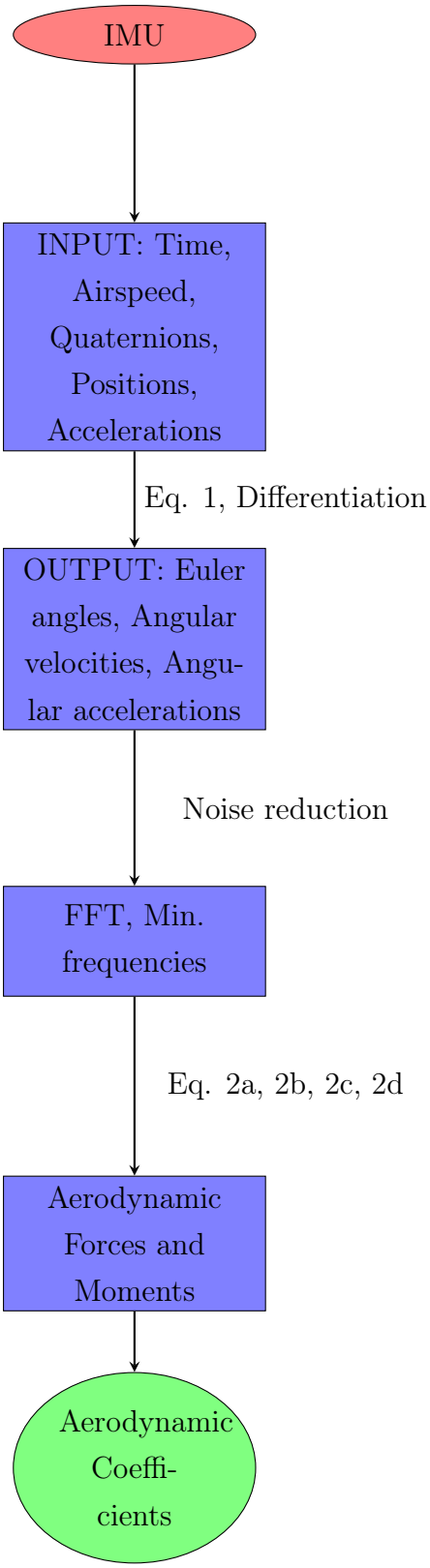
moments of inertia, which as previously seen are constant quantities. Therefore, it will result that the moment curves will be comparable and overlapping both macroscopically and, in large part, microscopically as those of the single angular velocities or accelerations. From these quantities it is finally possible to obtain the coefficients, through the well-known formulations that will be presented in the following chapter concerning the analysis of outdoor flight. Therefore, these quantities are not calculated in this chapter as it only has the function of comparing the quantities that influence these parameters.

4.2 Outdoor flight analysis

As seen in the previous chapter (REF COMPARISON), the comparison between the data detected by the OptiTrack camera system and the IMU mounted on board the aircraft revealed how it is fully reliable and, therefore, how it is possible to perform outdoor flights relying only on it. The purpose of this chapter is precisely to collect data from outdoor flights and study and understand the behaviour of the aircraft during the flight. The aerodynamic forces and moments and the relative coefficients will then be calculated. Furthermore, observing the manoeuvres performed by the aircraft, we will try to analyse these forces and moments in order to understand what the aircraft's limitations may be and, therefore, the improvements to be implemented on it in order to make it as efficient as possible.
The outdoor flight was performed at the model airplane run of the "Club Eole de Muret".

4.2.1 Data acquisition and processing

In order to avoid a repetition of the data processing already widely seen in the previous chapters, a flow diagram (4.2.1) is presented in this chapter which summarises the fundamental steps that characterize the analysis of outdoor flight. As visible, the steps are the same, except that in this case we will calculate and, subsequently, also analyse the aerodynamic forces, moments and coefficients of the aircraft. Consequently, to streamline the discussion, only the significant results for the analysis in question will be presented, leaving aside the problems already addressed.
So, just to summarise what will be done, we collect the data from the IMU on board the aircraft. The necessary data are: time, airspeed, quaternions, positions and linear accelerations. From the quaternions we obtain the Euler angles and the angular velocities and accelerations. The noise level in the data is reduced by using the previously used filter again. The aerodynamic forces and moments that characterize the behaviour of the aircraft are calculated. These forces and moments are non-dimensionalised to obtain the relative coefficients as the last result.



Thus, as already said, leaving out the data processing part, the first step is to graphically reproduce the flight performed by the aircraft. For this purpose, four overall views are reproduced (as done with the data coming from OptiTrack in the previous chapter): the first one in 3D, which again is coloured in order to distinguish the different phases of the flight, showing the overall flight and other

three in 2D to display the flight projected on two axes, thus reproducing the planes views XY, YZ and XZ. These reproductions can be seen in figures from 30 to 33.

As can be seen from the images, the flight can be divided into two basic parts:

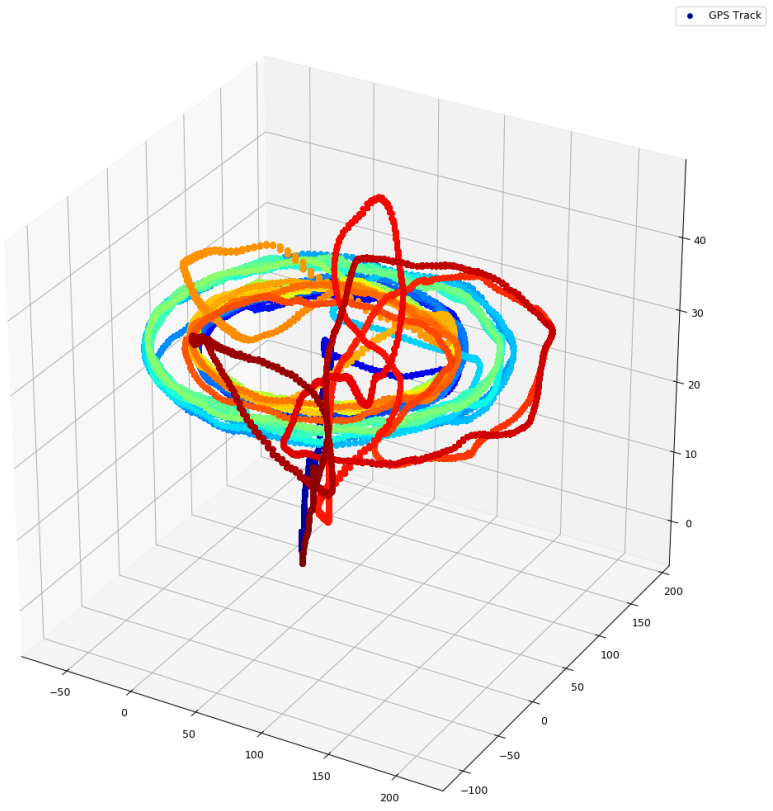


Figure 30: 3D reproduction of the flight track

a phase in autopilot mode, easily recognisable in the XY plane (fig. 31) since it is the one characterised by concentric circles, and a phase, instead, in manual flight, where the pilot controlled the aircraft remotely from the ground. It can be seen from the other two 2D views (YZ and XZ [fig. 32 and 33]) as in the automatic flight phase the aircraft maintains an altitude considerable as constant, while in the manual phase the altitude varies. These are fundamental aspects that will help us to better understand some behaviours of the aircraft if related to its parameters, e.g. speeds, accelerations, etc.

A first interesting idea is to compare the airspeed with the ground speed. The latter can be obtained by exploiting its 3 components (X, Y, Z) present in the array called speed collected from the IMU file. In fact it can be obtained from the following expression (eq. 3).

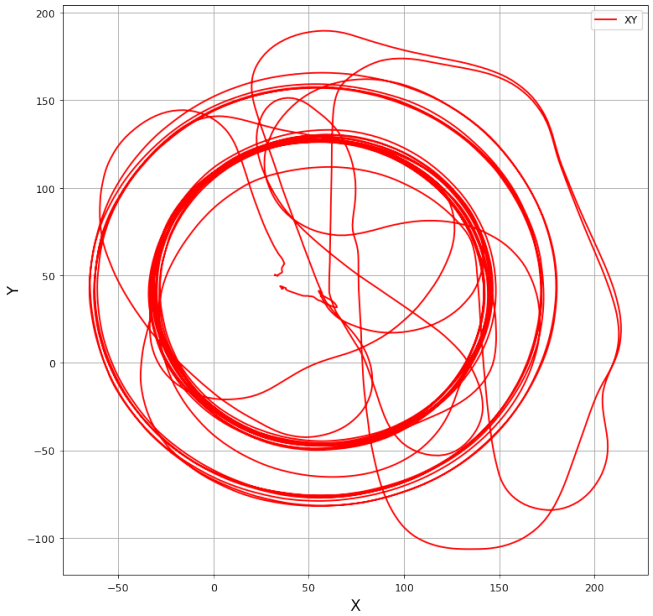


Figure 31: XY plane of the flight track

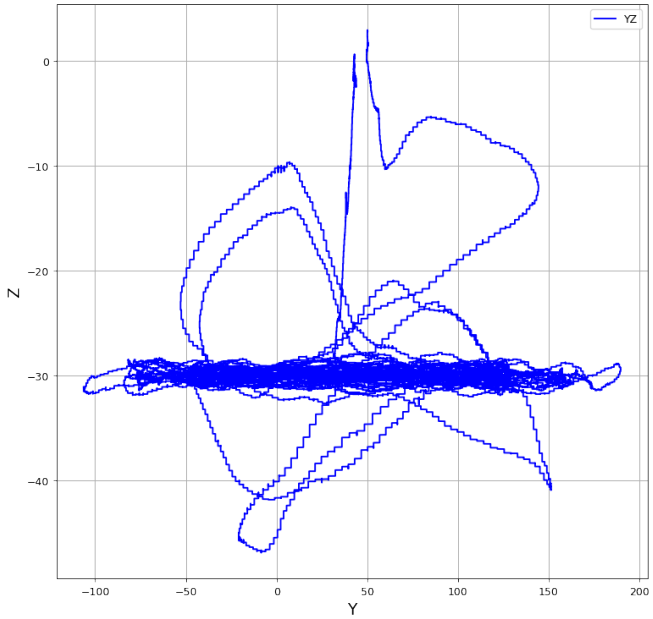


Figure 32: YZ plane of the flight track

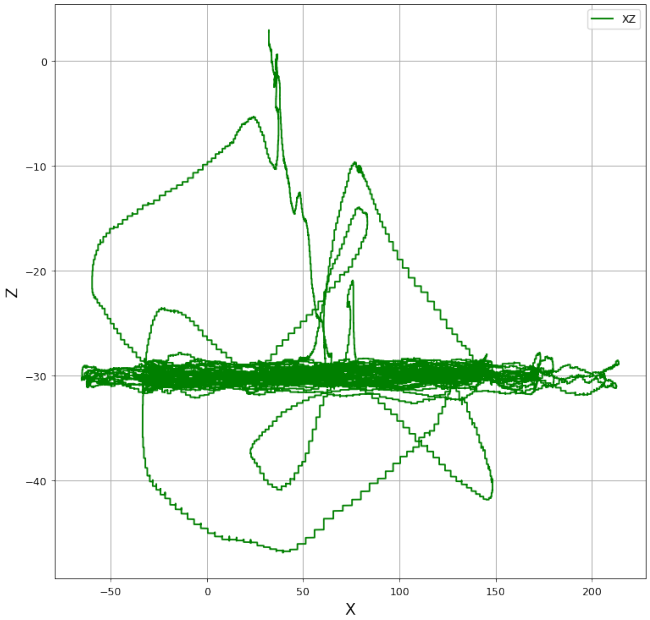


Figure 33: XZ plane of the flight track

$$GS = \sqrt{V_x^2 + V_y^2 + V_z^2} \tag{3}$$

The comparison between the two different speeds is shown in figure 34.

They have the same trend but different values point by point. This difference is

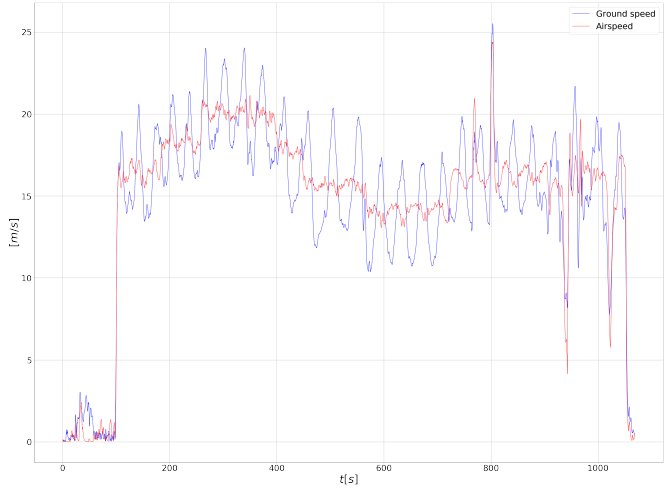


Figure 34: Comparison between the ground speed and the airspeed

due to the fact that, while the airspeed, as mentioned above, is the speed of the air flow that affects the aircraft and therefore the flight speed, the ground speed also takes into account the wind speed. Therefore, in the following paragraphs in which the coefficients deriving from the aerodynamic forces and moments will be calculated, the airspeed will be used, present in the denominator of the respective

expressions, and not the ground speed.

Lastly, we now calculate the fundamental angle that characterizes the flight: the angle of attack (35. It is the angle between the chord line of the wing of a fixed-wing aircraft and the vector representing the relative motion between the aircraft and the atmosphere.

To do this, trigonometric relationships applied to speeds are used. So, we first

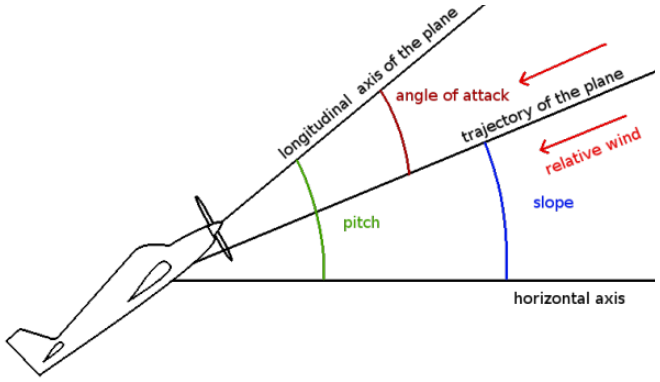


Figure 35: Representation of the angle of attack[15]

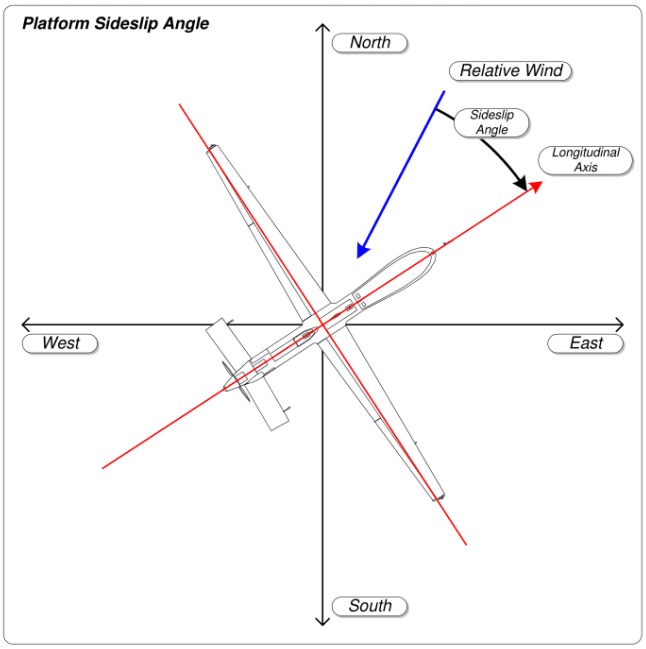


Figure 36: Representation of the sideslip angle[16]

calculate the value of γ, which is the angle between the horizontal plane and the trajectory of the aircraft

$$\gamma = atan(\frac{V_z}{V_x}) \tag{4}$$

and, then, using the pitch angle θ value , calculated from the quaternions coming from the IMU data, we proceed obtaining the angle of attack

$$\alpha = \theta - \gamma \tag{5}$$

Since through these calculations the slight noises previously eliminated of the respective variables in play have accumulated, before using these variables a short filtering is necessary, performed in the same way as previously presented. This angle has been calculated as it will be fundamental in order to calculate the lift and drag of the aircraft.
Theoretically, besides the angle of attack α, we should also consider the sideslip angle β (fig. 36), that is to say the rotation of the longitudinal axis of the aircraft with respect to the current flow that invests it. To take account of this angle, however, it would be advisable to mount a suitable instrument for the relative measurement, such as a magnetometer. Since, at the moment, the aircraft has no such instrument, it has been preferred to neglect the sideslip angle. However, in possible future configurations, the aircraft could be equipped with such a measurement tool that would enrich subsequent studies.

4.2.2 Calculation of forces and moments

At this point, as anticipated several times during the study, the next step is to calculate the aerodynamic forces and moments acting on the aircraft.
First, the forces acting on the aircraft are calculated. They are calculated by multiplying the mass of the aircraft by the accelerations on the three respective axes. A single precaution must be directed towards the force acting on the Z axis. In fact, it is known how gravitational acceleration g also acts on that axis. Consequently, this phenomenon must be taken into account. Therefore, these forces are derived from the following expression,

$$F = [a_x \; a_y \; (a_z + g)] \; m \tag{6}$$

As for the aerodynamic forces of interest for this research, they are calculated using the projections of the newly calculated forces. In fact, it could be wrongly

thought that the lift is the vertical force and the drag the horizontal force. In reality by definition, they are respectively the forces perpendicular and parallel (and opposite) to the direction of motion. Therefore they can have deviation angles with respect to the vertical and horizontal axes that cannot be neglected. They are then calculated using the following expressions, found in literature [8],

$$L = -F_z \ cos\alpha + F_x \ sin\alpha \tag{7a}$$

$$D = -F_z \ sin\alpha - F_x \ cos\alpha \tag{7b}$$

As shown in the two equations (7a and 7b), they present the contributions of two different forces, confirming what was said above. The graphs representing the trend of these forces are presented in figures 37 and 38.

As regards the calculation of aerodynamic moments, they have a more com-

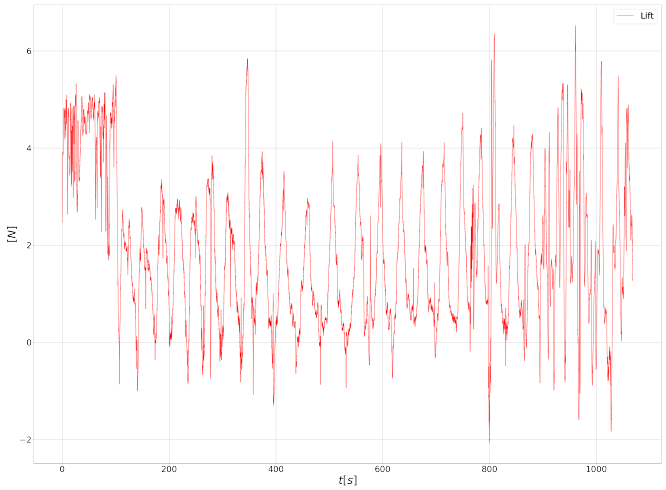


Figure 37: Lift for the outdoor flight

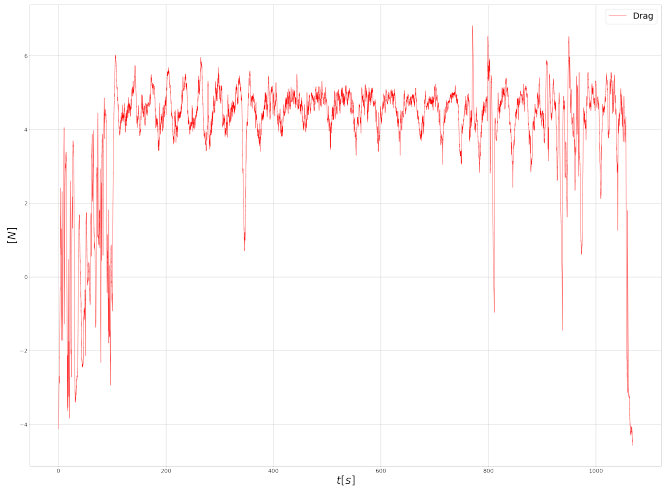


Figure 38: Drag for the outdoor flight

plicated formulation than that of forces, as found in the literature. In fact, to calculate these moments generated by the aerodynamic loads acting on the plane, the following method is used. Starting from the moments of inertia and the angular rates of the aircraft (p, q, r), the aerodynamic moments are found using the rotational equation of motion,

$$\mathbf{M} = \frac{d(I\omega)}{dt} \tag{8}$$

where I and ω are a general way to represent the moment of inertia and the angular velocity. The moments of inertia are those shown in table 1 and are assumed to be constant. Moreover, the standard assumption to ignore the crossed moments of inertia (I_{xy}, I_{yz}, I_{xz}) is used, since these terms are very small and the aircraft is supposed to be symmetric about these axis. The final expressions for the three moments of interest are, then, the following.

$$M_x = \dot{p}I_{xx} + qr(I_{zz} - I_{yy}) \tag{9a}$$

$$M_y = \dot{q}I_{yy} + pr(I_{xx} - I_{zz}) \tag{9b}$$

$$M_z = \dot{r}I_{zz} + pq(I_{yy} - I_{xx}) \tag{9c}$$

These three equations represent, in order, the rolling, pitching and yawing moments. They are important to understand the response of the airplane. The results of these calculations are shown in figures from 39 to 41.

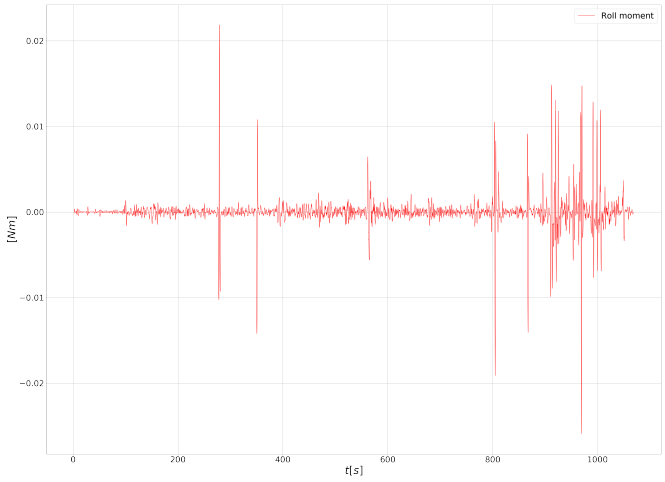


Figure 39: Rolling moment for outdoor flight

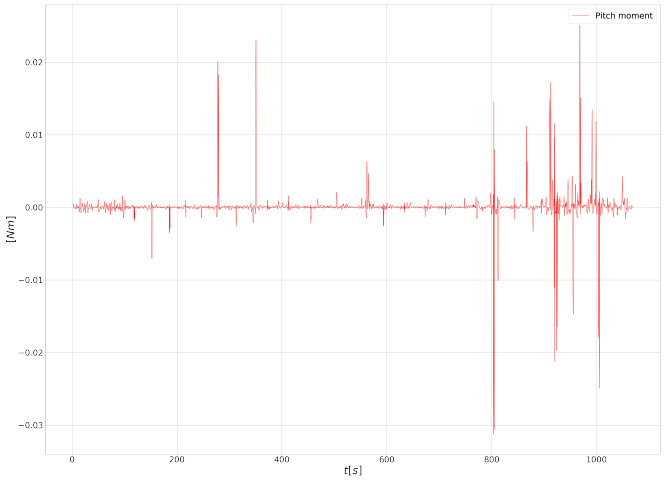


Figure 40: Pitching moment for outdoor flight

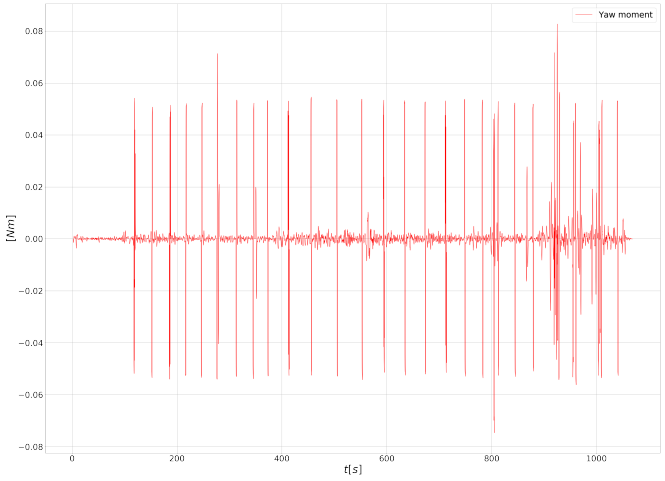


Figure 41: yawing moment for outdoor flight

4.2.3 Calculation of coefficients

There is now nothing left but to calculate the coefficients deriving from these forces and moments, which represent their dimensionless form and therefore, in a certain sense, generalised as independent of the geometry and the conditions of the surrounding environment.
The coefficients of lift and drag are then calculated first. Remembering the general expressions for the lift and the drag,

$$L = \frac{1}{2}\rho V^2 S c_L \tag{10a}$$

$$D = \frac{1}{2}\rho V^2 S c_D \tag{10b}$$

it is easily possible to derive the relative coefficient, making it explicit from these two equations. The expressions used are, then, the following,

$$c_L = \frac{2L}{\rho V^2 S} \tag{11}$$

$$c_D = \frac{2D}{\rho V^2 S} \tag{12}$$

where ρ represents the density, S represents the wet surface, that is twice the wing surface as it is necessary to consider both the back and the belly of the wing, and V represents the airspeed, as mentioned and explained before (4.2.1).
At this point, the same study can be repeated for the coefficients relating to aerodynamic moments. Knowing therefore the general expression of the moment

$$M = \frac{1}{2}\rho V^2 S c_M c \tag{13}$$

the coefficient can be obtained analogously to what was done before, or inverting this expression. The following formulation is then obtained

$$c_M = \frac{2M}{\rho V^2 S c} \tag{14}$$

They present approximately the same expression, with the only difference that in this case the mean chord of the wing is also present as a denominator, since as it is known a moment is a force multiplied by a distance.
In this case, the moments analysed are three, one for each axis (X, Y, Z) and, therefore, the three respective formulations will be the following,

$$c_{M_x} = \frac{2M_x}{\rho V^2 S c} \tag{15a}$$

$$c_{M_y} = \frac{2M_y}{\rho V^2 S c} \tag{15b}$$

$$c_{M_z} = \frac{2M_z}{\rho V^2 S c} \tag{15c}$$

After having calculated all these coefficients, it is possible, as previously done with the other quantities, to plot them in graphs to better understand their behaviour. These graphs can be viewed in figures from 42 to 46.

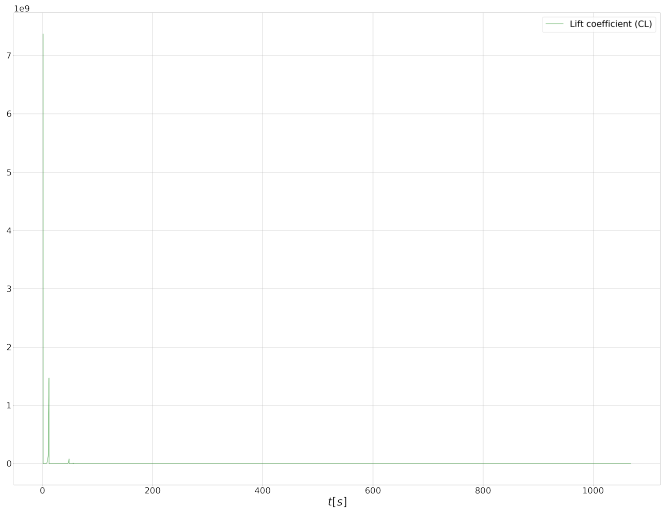


Figure 42: Lift coefficient

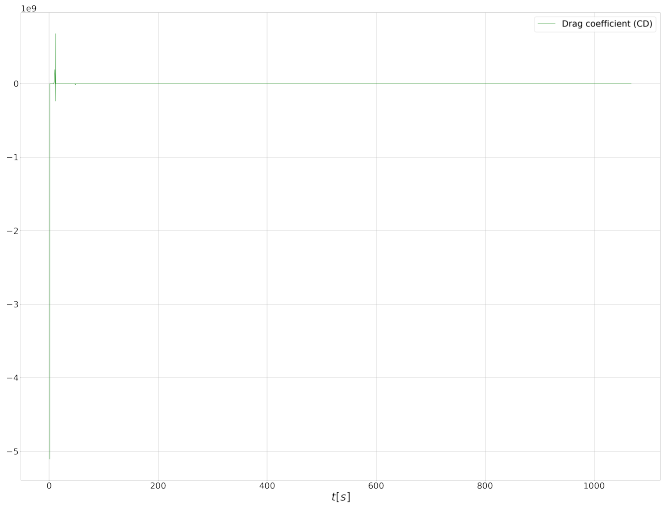


Figure 43: Drag coefficient

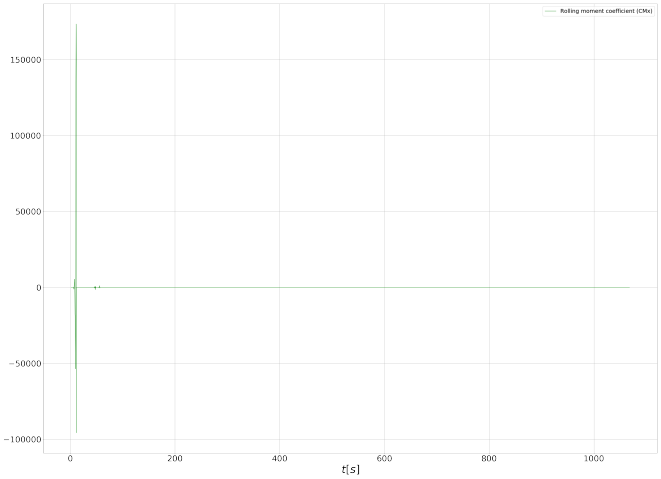


Figure 44: Rolling moment coefficient

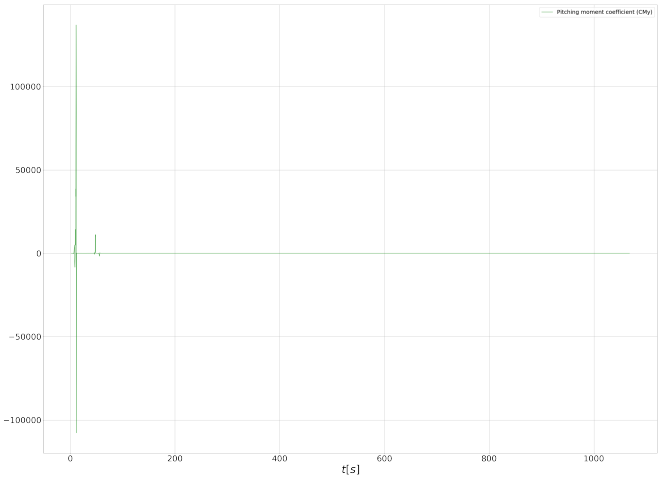


Figure 45: Pitching moment coefficient

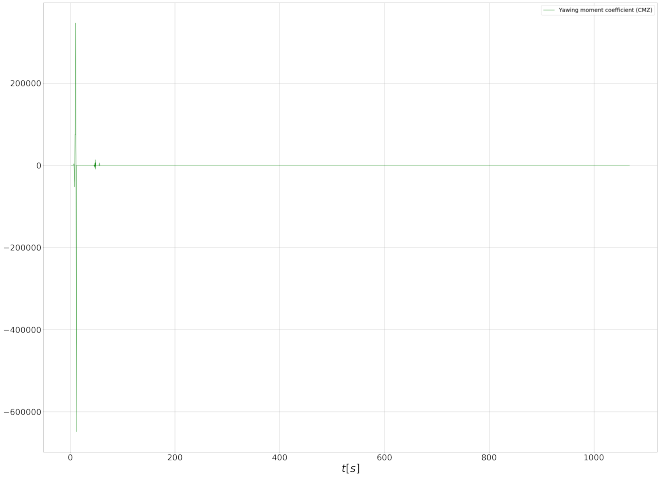


Figure 46: Yawing moment coefficient

The first thing we notice are the exaggerated values for the coefficients. In fact, the coefficients of lift and drag are of the order of 10^9, when they should be around the unit. While those of the moments, instead, are around hundreds of thousands. This problem is due to the fact that in the initial and final flight phase the IMU recorded the parameters when the aircraft was still held by the pilot. Therefore the initial and final sections of the graphs are not to be taken into consideration. In fact, if the same graphics are isolated from the actual flight phase alone, there are completely different results, visible in figures 47 to 51.

In fact, we can now see that the values make much more sense. It is enough

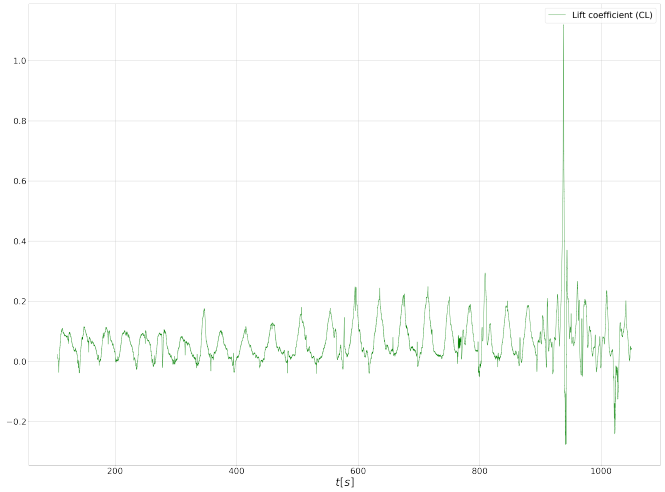


Figure 47: Lift coefficient for actual flight

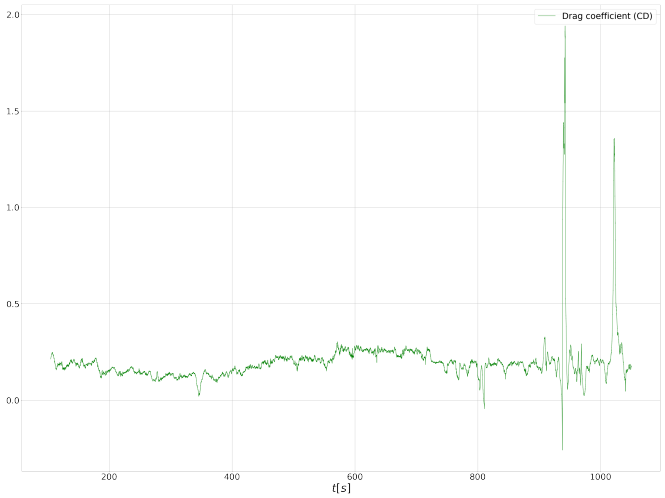


Figure 48: Drag coefficient for actual flight

to pay attention, for example, to the graph of the coefficient of lift (figure 47). In this case, as mentioned above, the values are lower than unity and, although they may not be the best prospect for a coefficient of importance, they represent

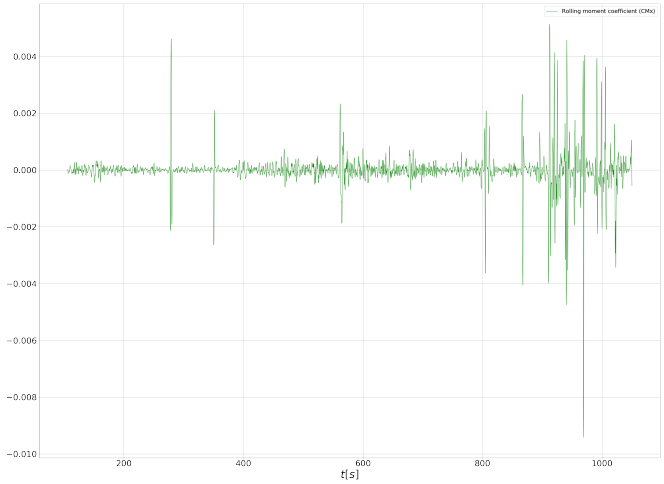


Figure 49: Rolling moment coefficient for actual flight

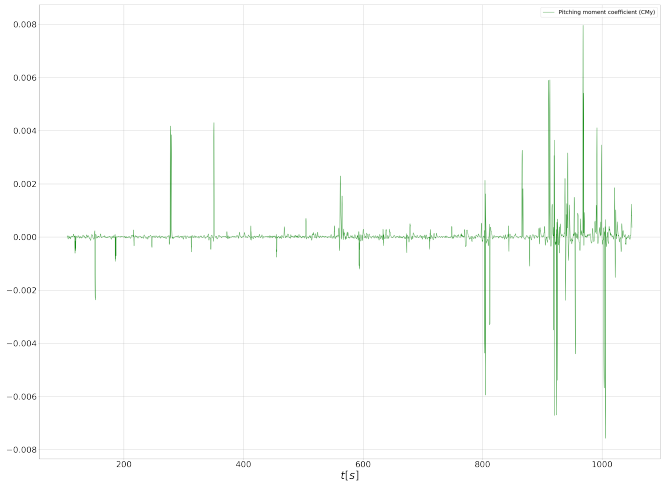


Figure 50: Pitching moment coefficient for actual flight

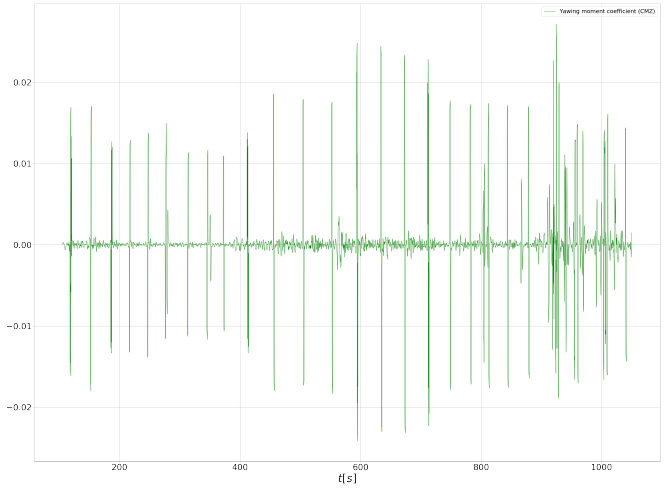


Figure 51: Yawing moment coefficient for actual flight

much more common and sensible values.

4.2.4 Results and discussion

First, we analyse the lift-to-drag ratio of the aircraft,

$$E = \frac{L}{D} = \frac{C_L}{C_D} \tag{16}$$

so as to relate the coefficient of lift with that of drag. As already seen for the single coefficient diagrams, the one related to L/D ratio assumes disproportionate values outside the horizontal flight phase of the aircraft. For this reason the graph is shown in the first place after having isolated the horizontal flight part (fig. 52), in which through a short "for" cycle the too high values corresponding to the points where the C_D touches zero have been shut down, values that bring the L/D ratio to values too high to understand its real physical sense.

As can therefore be seen from this graph, the L/D ratio assumes values for the

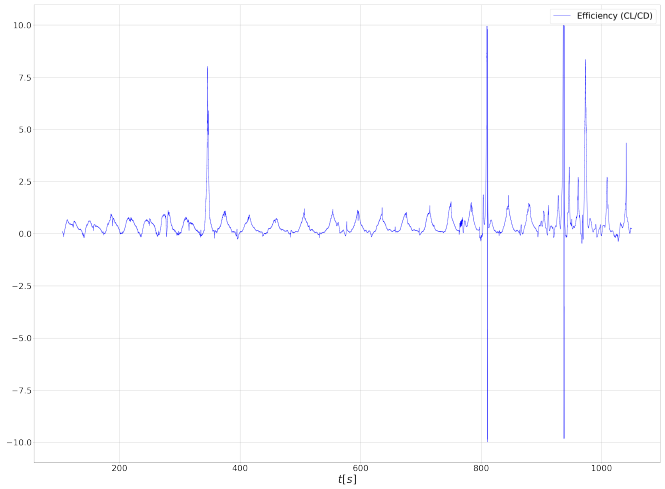


Figure 52: Lift-to-drag ratio for actual flight

most part between 0 and 1 or slightly higher than unity. This translates into a slight preponderance of drag against lift. In fact, as shown in the lift and drag graphs seen in the previous chapter (fig. 37 and 38), the lift assumed values visibly higher than those of the drag only in the initial phase of the flight, that is to say in that in which the aircraft must take altitude to move to the desired configuration in which to operate then the horizontal flight, phase in which instead the drag assumes oscillating values around zero. This, aerodynamically speaking, therefore translates into two ways: the parasitic drag and the induced drag are very high. The first depends on the viscous friction and the shape of the aircraft, factors that can therefore be improved by acting on the design phase of the aircraft,

looking for geometries that have less influence on this parameter. The second one, on the other hand, depends on the tips vortexes that are created due to the back-belly communication. This parameter could be decreased, for example, by increasing the aircraft's aspect ratio (AR), which causes not only a decrease in the drag coefficient, according to the expression

$$C_{D_i} = \frac{C_L^2}{\pi e AR} \tag{17}$$

but also an increase in the lift coefficient, according to

$$C_L = \frac{C_{L_\alpha}\alpha}{1 + \frac{C_{L_\alpha}}{\pi AR}} \tag{18}$$

Obviously the contribution due to the wave drag is completely neglected because the aircraft operates at speeds totally outside the range in which the transonic phenomena begin to appear.

Another very important parameter for the aerodynamic analysis, which has not yet been discussed as irrelevant in the preceding discussion, is the lateral force F_y. In fact this gives us an idea of the lateral stability of the aircraft. If we analyse the relative graph, visible in figure 53, we see how this is not null, but oscillates around zero assuming sometimes even important values, therefore it cannot be neglected. If we consider for example a small section in which this

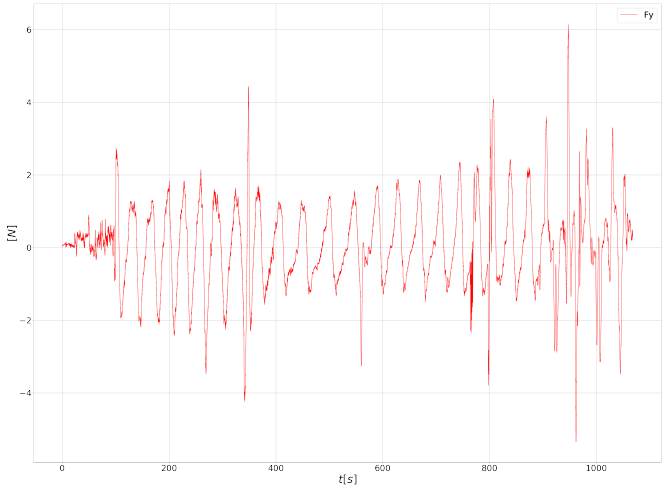


Figure 53: Lateral force for actual flight

force assumes non-negligible values, such as the one in figure 54 in which the value of this force also changes its sign, and we compare it with the relative part of the 2D visualisation of the flight on the XY plane (fig. 55), we see how the trajectory of the aircraft does not present deviations worthy of note. This means

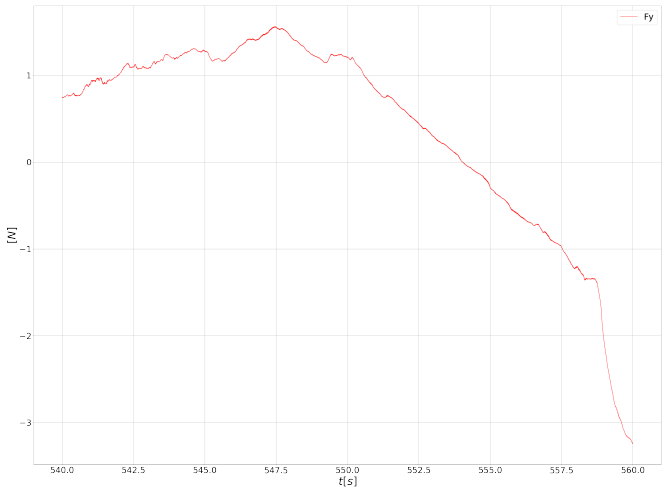


Figure 54: Zoom on lateral force

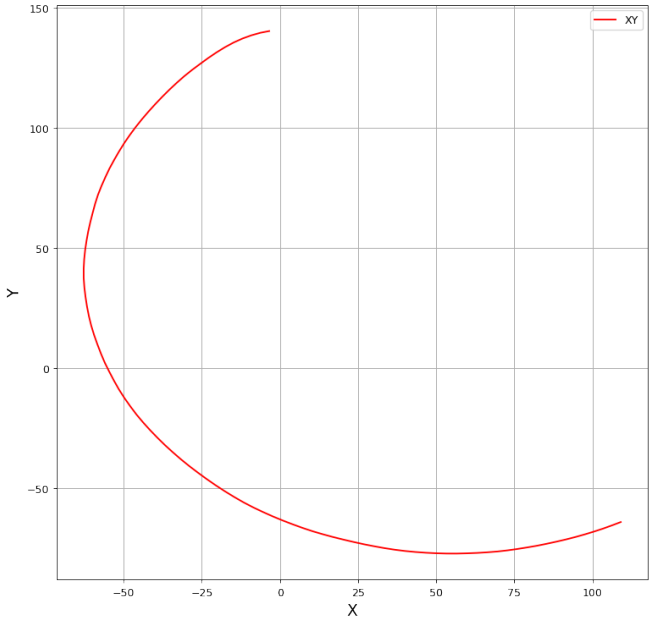


Figure 55: Zoom on XY plane

that through the instruments mounted on board, the aircraft recognises the magnitude of the lateral force acting on it and is therefore able to counter it without causing it to compromise the flight, thus keeping its stability intact. From this study, it is therefore not necessary for future versions to improve this aspect as it is already optimal at present.

A question that can now be asked is: if it were not possible to obtain 2D or 3D visualisations, could we identify the different phases of the flight from the parameters that characterize them? For example, can we distinguish internal circles from external circles (fig. 31) thanks to speed or lift? Let's take the 2D view on the XY plane in which we isolate the two flight phases mentioned, or those related to the inner and outer circles in a clockwise direction (fig. 56). Let's draw the relative graph of the lift (fig. 57). It is clearly visible how the differences are almost invisible between one phase and another and therefore we cannot use the lift. If, on the other hand, the coefficient of lift is examined (fig. 58), a slight difference in the values assumed in the two cases is begun, but this difference is still too slight to be considered discriminating between the two situations. Then

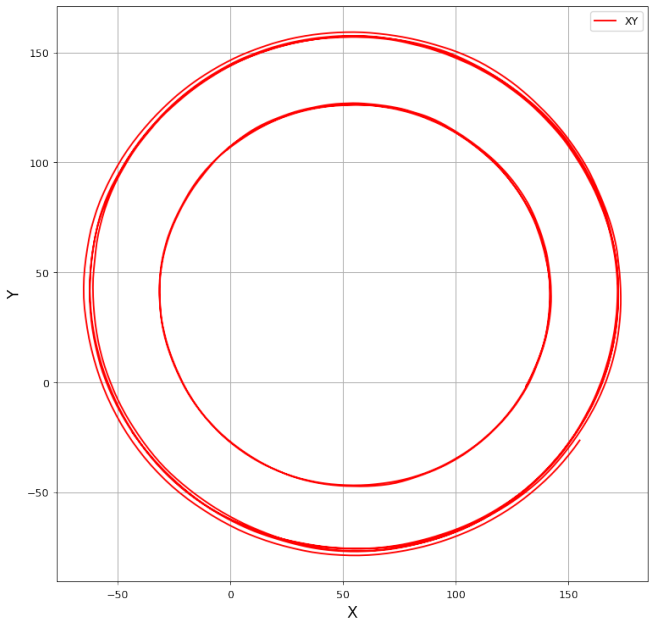


Figure 56: Clockwise inner and outer circles on XY plane

we analyse the airspeed. In the relative graph (fig. 59) the left part identifies the outer circles, while the right one identifies the internal ones. In this case, the two phases are marked by clearly different speed values. In fact, as perhaps we can guess, the inner circles are characterized by lower speeds that allow, therefore, to travel smaller curvature radii in the turn. In fact, if the speed were too high, an excessively high centripetal acceleration would be produced which would threaten

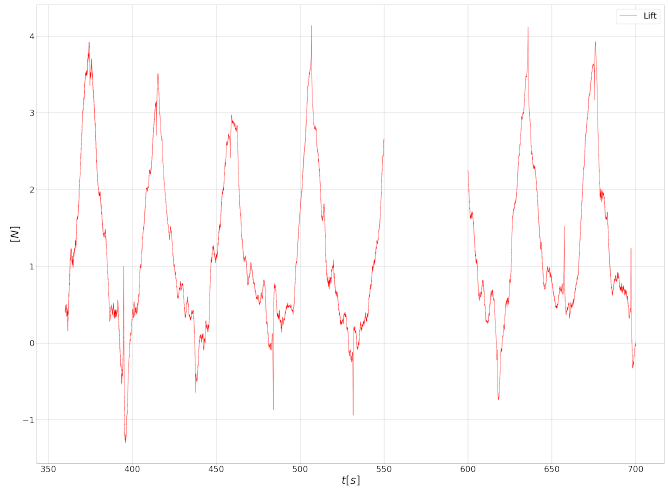


Figure 57: Lift for clockwise inner and outer circles

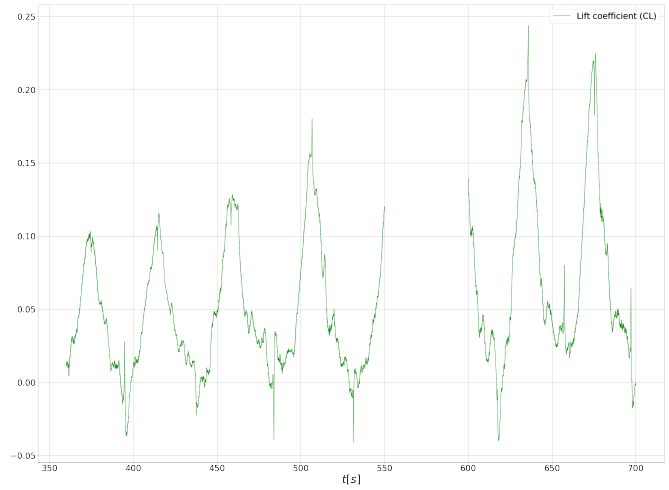


Figure 58: Lift coefficient for clockwise inner and outer circles

the dynamic stability of the aircraft, a stability which instead, as stated before, is optimal for this aircraft. Is there, instead, any difference in these parameters

Figure 59: Airspeed for clockwise inner and outer circles

between clockwise and anti-clockwise circles? In this case, again, the lift and the relative coefficient do not come to our aid, as they present really too slight differences to be able to draw a conclusion. As for the speed analysis, it is more useful to consider the angular velocities. In fact, both the roll rate (fig. 60) and

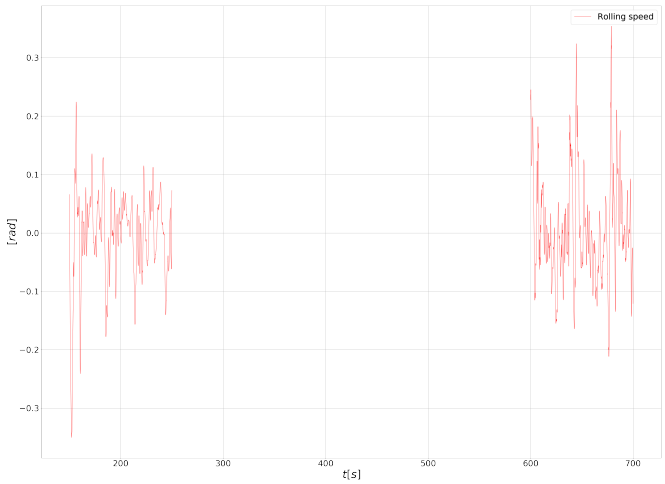


Figure 60: Rolling speed for counterclockwise and clockwise inner circles

the yaw speed (fig. 61) have peaks of opposite sign in the two cases. In fact, as imaginable and as visible in the two graphs, the orientations of these two speeds are opposite, orientation clearly recognisable in the peaks, especially in the yaw speed. This does not happen, however, in the pitching speed. In fact, it takes on the same sign whether you are on a left turn or a right turn.

Finally, as already mentioned when the graphs of the aerodynamic forces, moments and coefficients were presented, the initial and final parts of them are

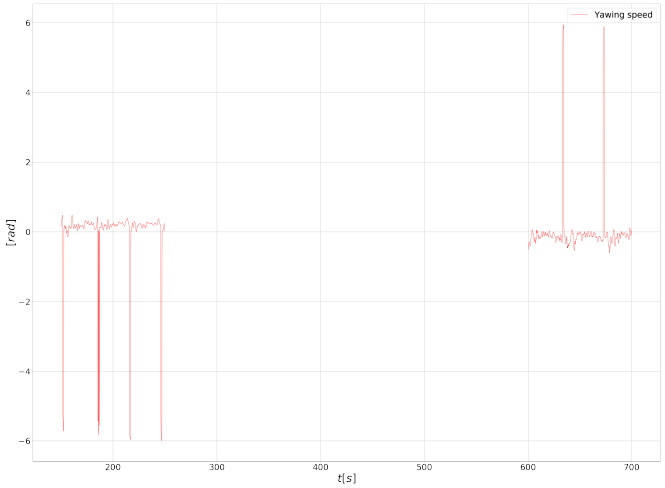


Figure 61: Yawing speed for counterclockwise and clockwise inner circles

irrelevant to the study of the aircraft's performance. In fact, they represent only the moment in which it is in the hands of the pilot and the period of adjustment from it, useless therefore to the global study. Therefore, from this it can be understood how in future configurations it may be useful to implement the aircraft with a landing gear or with any type of element that allows autonomous support. In this way, human action is avoided and it is possible to analyze a complete flight from take-off to landing. The latter could be useful as landing tests can also be carried out in sensitive areas, thus defining DarkO's action limits.

5 Conclusions

This study has tried, first of all, to understand if in the future it is possible to study the performance of a small VTOL aircraft using only the data coming from the instruments mounted on board, in particular the IMU. To do this, these data were initially compared with those from an infrared camera system (OptiTrack) that could be considered exact, therefore flying in a closed space isolated from the outside. In this way, comparing the speeds and the accelerations coming from the two systems it was found that they are totally comparable and, therefore, that the IMU mounted on board the aircraft is a more than reliable tool for the study and analysis of flights outdoor, flights where it is possible to have a complete picture of the phenomena that can affect the performance of the aircraft. This is not only a step forward from an analytical point of view, but also from the point of view of the accessibility of such studies. In fact, it is not always possible to have access to sophisticated analysis tools, such as wind tunnels, therefore it has also been shown how it is possible to conduct a study of the performance of an aircraft using tools accessible to anyone as they are on the market at prices very low.

Subsequently, to demonstrate what has just been said, a flight performed externally was analysed, in order to show how it is possible to extrapolate any type of data from those provided by the IMU. We also briefly analysed the data obtained, focusing on the most significant and more direct data for the reader.

Therefore, in conclusion, a solution was provided for those problems listed and addressed in the introduction of this research and also found by similar studies on other aircraft models. Furthermore, by analysing the outdoor flight it was also possible to understand what could be the improvements to be implemented in future versions of the aircraft taken as an example, as it is characterized by continuous updating.

References

[1] Krossblade Aerospace. *VTOL - VERTICAL TAKE OFF AND LANDING*. URL: https://www.krossblade.com/vtol-vertical-take-off-and-landing.

[2] G. Casiez, N. Roussel, and D. Vogel. "1€ Filter: A Simple Speed-based Low-pass Filter for Noisy Input in Interactive Systems. Conference on Human Factors in Computing Systems - Proceedings". In: (2012). DOI: 10.1145/2207676.2208639.

[3] FEHR&PEERS. *Vertical Take-Off & Landing: A New Form of Mobility?* URL: https://www.fehrandpeers.com/vtol/.

[4] K. Gryte et al. "Aerodynamic modeling of the Skywalker X8 Fixed-Wing Unmanned Aerial Vehicle". In: (2018). DOI: 10.1109/ICUAS.2018.8453370.

[5] Paragraph entirely taken from Jacson et. al. "Model-Free Control for Hybrid Micro Air Vehicles". On-going Ph. D.

[6] Z. R. Manchester et al. "A Variable Forward-Sweep Wing Design for Improved Perching in Micro Aerial Vehicles". In: (2017). DOI: 10.2514/6.2017-0011.

[7] OptiTrack. *OptiTrack - Motion Capture System*. URL: http://www-cs-faculty.stanford.edu/~uno/abcde.html.

[8] D V Uhlig et al. "Determining Aerodynamic Charateristics of a Micro Air Vehicle Using Motion Tracking". In: *American Institute of Aeronautics and Astronautics* 2010-8416 (2010). DOI: 10.2514/6.2010-8416.

[9] Vicon. *Vicon MX System*. URL: http://www.vicon.com/products/viconmx.html.

[10] Wikipedia. *VTOL*. URL: https://it.wikipedia.org/wiki/VTOL.

A Appendix A - Linear Speeds and Accelerations Comparison

```
#!/usr/bin/env python
from __future__ import print_function, division
import numpy as np
import math
import pandas as pd
import re
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import seaborn #plotting lib, but just adding makes the
    matplotlib plots better
import glob # use this for file O later
from numpy import sin, cos, pi, sqrt, dot
from scipy import stats, optimize
from scipy import linalg as la
from scipy.optimize import leastsq
from scipy.optimize import least_squares
import pdb
from mpl_toolkits.mplot3d import Axes3D
from scipy.interpolate import griddata, interp1d
from scipy import signal
from scipy.optimize import curve_fit
from scipy import interpolate
import scipy as sp

##FILTRO PASSABASSO

class LowPassFilter(object):

    def __init__(self, alpha):
        self.__setAlpha(alpha)
        self.__y = self.__s = None
```

```
    def __setAlpha(self, alpha):
        alpha = float(alpha)
        if alpha<=0 or alpha>1.0:
            raise ValueError("alpha (%s) should be in (0.0, 1.0]"%alpha)
        self.__alpha = alpha

    def __call__(self, value, timestamp=None, alpha=None):
        if alpha is not None:
            self.__setAlpha(alpha)
        if self.__y is None:
            s = value
        else:
            s = self.__alpha*value + (1.0-self.__alpha)*self.__s
        self.__y = value
        self.__s = s
        return s

    def lastValue(self):
        return self.__y

##ONEEUROFILTER FOR NOISE FILTERING

class OneEuroFilter(object):

    def __init__(self, freq, mincutoff=1.0, beta=0.0, dcutoff=1.0):
        if freq<=0:
            raise ValueError("freq should be >0")
        if mincutoff<=0:
            raise ValueError("mincutoff should be >0")
        if dcutoff<=0:
            raise ValueError("dcutoff should be >0")
        self.__freq = float(freq)
        self.__mincutoff = float(mincutoff)
        self.__beta = float(beta)
        self.__dcutoff = float(dcutoff)
```

```
        self.__x = LowPassFilter(self.__alpha(self.
    __mincutoff))
        self.__dx = LowPassFilter(self.__alpha(self.
    __dcutoff))
        self.__lasttime = None

    def __alpha(self, cutoff):
        te    = 1.0 / self.__freq
        tau   = 1.0 / (2*math.pi*cutoff)
        return  1.0 / (1.0 + tau/te)

    def __call__(self, x, timestamp=None):
        # ---- update the sampling frequency based on
    timestamps
        if self.__lasttime and timestamp:
            self.__freq = 1.0 / (timestamp-self.__lasttime
    )
        self.__lasttime = timestamp
        # ---- estimate the current variation per second
        prev_x = self.__x.lastValue()
        dx = 0.0 if prev_x is None else (x-prev_x)*self.
    __freq # FIXME: 0.0 or value?
        edx = self.__dx(dx, timestamp, alpha=self.__alpha(
    self.__dcutoff))
        # ---- use it to update the cutoff frequency
        cutoff = self.__mincutoff + self.__beta*math.fabs(
    edx)
        # ---- filter the given value
        return self.__x(x, timestamp, alpha=self.__alpha(
    cutoff))

def Euro_filter(signal,config):
    f = OneEuroFilter(**config)
    timestamp = 0.0 # seconds
    filtered = np.zeros([int(len(signal))])
    for i in range(len(signal)):
        filtered[i] = f(signal[i], timestamp)
        timestamp += 1.0/config["freq"]
    return filtered
```

```
# READ THE SD CARD

file_name = "pprzlog_0088.LOG"; # Carica il file dati registrati sulla SD a bordo

dt = np.dtype('i4');
di = np.fromfile(file_name, dtype=dt);

dt = np.dtype('f4');
df = np.fromfile(file_name, dtype=dt);

col = 12+26;
exc = np.mod(di.size,col);
row = int((di.size-exc)/col);

di = di[:-exc].reshape(row,col);
df = df[:-exc].reshape(row,col);

d = np.zeros([row,col]);
d[:,0:12]  = di[:,0:12];
d[:,12:]   = df[:,12:];
data = d.copy()

# Sample frequency
sf = 500;
fo_c = 0.025 #First order actuator dynamics constant (discrete, depending on sf)
N = data.shape[0]
t = np.arange(N)/sf
counter = data[:,0]
airspeed = data[:,11]/2**19
quat = data[:,15:19]
pos = data[:,19:22]
speed = data[:,22:25]
accel_ned = data[:,32:35]

config = {
    'freq': 500,        # Hz
```

```
    'mincutoff': 0.5,  # 1.0 FIXME
    'beta': 0.01,       # 1.0 FIXME
    'dcutoff': 0.5     # 1.0 this one should be ok
    }
vx_sd = Euro_filter(speed[:,0],config)
config = {
    'freq': 500,       # Hz
    'mincutoff': 0.5,  # 1.0 FIXME
    'beta': 0.01,       # 1.0 FIXME
    'dcutoff': 0.5     # 1.0 this one should be ok
    }
vy_sd = Euro_filter(speed[:,1],config)
config = {
    'freq': 500,       # Hz
    'mincutoff': 0.5,  # 1.0 FIXME
    'beta': 0.01,       # 1.0 FIXME
    'dcutoff': 0.5     # 1.0 this one should be ok
    }
vz_sd = Euro_filter(speed[:,2],config)
config = {
    'freq': 500,       # Hz
    'mincutoff': 0.5,  # 1.0 FIXME
    'beta': 0.01,       # 1.0 FIXME
    'dcutoff': 0.5     # 1.0 this one should be ok
    }
v_sd = Euro_filter(airspeed,config)
config = {
    'freq': 500,       # Hz
    'mincutoff': 0.1,  # 1.0 FIXME
    'beta': 0.01,       # 1.0 FIXME
    'dcutoff': 0.5     # 1.0 this one should be ok
    }
ax_sd = Euro_filter(accel_ned[:,0],config)
config = {
    'freq': 500,       # Hz
    'mincutoff': 0.1,  # 1.0 FIXME
    'beta': 0.01,       # 1.0 FIXME
    'dcutoff': 0.5     # 1.0 this one should be ok
    }
```

```
ay_sd = Euro_filter(accel_ned[:,1],config)
config = {
    'freq': 500,        # Hz
    'mincutoff': 0.1,   # 1.0 FIXME
    'beta': 0.008,        # 1.0 FIXME
    'dcutoff': 0.5      # 1.0 this one should be ok
    }
az_sd = Euro_filter(accel_ned[:,2],config)

fig1 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
matplotlib.rc('xtick', labelsize=20)
matplotlib.rc('ytick', labelsize=20)
plt.plot(speed[:,0], linewidth='1',label='Vx IMU Not
   filtered')
plt.plot(vx_sd, linewidth='1',label='Vx IMU Filtered') #
   smooth by filter
#plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[m/s]$',fontsize=40)
plt.legend(fontsize='30')
fig2 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
plt.plot(speed[:,1], linewidth='1',label='Vy IMU Not
   filtered')
plt.plot(vy_sd, linewidth='1',label='Vy IMU Filtered') #
   smooth by filter
#plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[m/s]$',fontsize=40)
plt.legend(fontsize='30')
fig3 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
plt.plot(speed[:,2], linewidth='1',label='Vz IMU Not
   filtered')
plt.plot(vz_sd, linewidth='1',label='Vz IMU Filtered') #
   smooth by filter
#plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[m/s]$',fontsize=40)
plt.legend(fontsize='30')
fig3 = plt.figure(figsize=(40, 30), dpi=60)
```

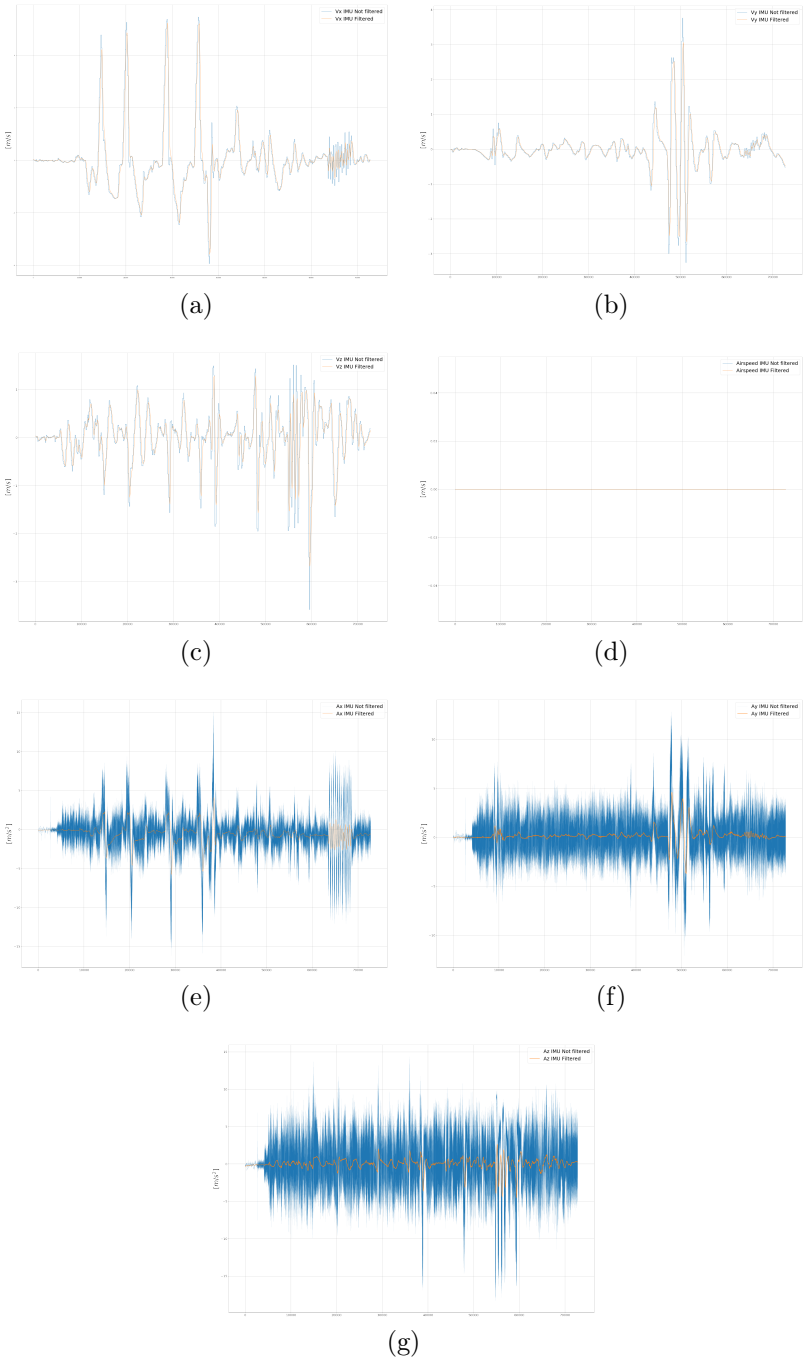
```
plt.grid()
plt.plot(airspeed, linewidth='1',label='Airspeed IMU Not
    filtered')
plt.plot(v_sd, linewidth='1',label='Airspeed IMU Filtered'
    ) # smooth by filter
#plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[m/s]$',fontsize=40)
plt.legend(fontsize='30')
fig4 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
matplotlib.rc('xtick', labelsize=20)
matplotlib.rc('ytick', labelsize=20)
plt.plot(accel_ned[:,0], linewidth='0.1',label='Ax IMU Not
    filtered')
plt.plot(ax_sd, linewidth='1',label='Ax IMU Filtered') #
    smooth by filter
#plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[m/s^2]$',fontsize=40)
plt.legend(fontsize='30')
fig4 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
plt.plot(accel_ned[:,1], linewidth='0.1',label='Ay IMU Not
    filtered')
plt.plot(ay_sd, linewidth='2',label='Ay IMU Filtered') #
    smooth by filter
#plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[m/s^2]$',fontsize=40)
plt.legend(fontsize='30')
fig4 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
plt.plot(accel_ned[:,2], linewidth='0.1',label='Az IMU Not
    filtered')
plt.plot(az_sd, linewidth='2',label='Az IMU Filtered') #
    smooth by filter
#plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[m/s^2]$',fontsize=40)
plt.legend(fontsize='30')

# READ THE OPTITRACK FILE
```

```
df_opti = pd.read_csv('DarkO1_15_03_2019_voliere.csv',
    skiprows=6)
df_opti.drop(['Frame'],axis=1,inplace=True)
sf = 1000
st = 0*sf; fn = 145*sf;

Px = -df_opti[st:fn].PZ
Py = df_opti[st:fn].PX
Pz = -df_opti[st:fn].PY

diff_period = sf
VX = Px.diff(periods=diff_period)
VY = Py.diff(periods=diff_period)
VZ = Pz.diff(periods=diff_period)

AX = VX.diff(periods=diff_period)
AY = VY.diff(periods=diff_period)
AZ = VZ.diff(periods=diff_period)

fig = plt.figure(figsize=(15,15), dpi=90)
ax = plt.axes(projection='3d')
# ax = fig.add_subplot(4,2,7, projection='3d');
#ax.plot3D(PX,PY,PZ, 'green', label='Ground Track');plt.
    legend();
cm = plt.cm.jet(np.linspace(0,1,(len(Px))))
ax.scatter(Px,Py,Pz, "-", c=cm, label='Flight Track');
plt.legend();

fig = plt.figure(figsize=(10,10), dpi=90)
matplotlib.rc('xtick', labelsize=10)
matplotlib.rc('ytick', labelsize=10)
plt.plot(Px,Py, 'r',label='XY')
plt.xlabel('X',fontsize=20)
plt.ylabel('Y',fontsize=20)
plt.grid()
plt.legend()
plt.show()
```

```
1275
fig = plt.figure(figsize=(10,10), dpi=90)
plt.plot(Py,Pz, 'b',label='YZ')
plt.xlabel('Y',fontsize=20)
plt.ylabel('Z',fontsize=20)
1280 plt.grid()
plt.legend()
plt.show()

fig = plt.figure(figsize=(10,10), dpi=90)
1285 plt.plot(Px,Pz, 'g',label='XZ')
plt.xlabel('X',fontsize=20)
plt.ylabel('Z',fontsize=20)
plt.grid()
plt.legend(loc='upper left')
1290 plt.show()

fig1 = plt.figure(figsize=(40, 30), dpi=90)
plt.grid()
matplotlib.rc('xtick', labelsize=30)
1295 matplotlib.rc('ytick', labelsize=30)
sf = 1000 # Need to Check this one !!!!!
st = 0*sf; fn = 145*sf;
N = df_opti.shape[0]
t = np.arange(N)/sf
1300 plt.plot(t[st:fn],AX[st:fn],linewidth='1',label='X axis
    acceleration OptiTrack')
sf = 500
st = 0*sf; fn = 145*sf;
N = data.shape[0]
t = np.arange(N)/sf
1305 plt.plot(t[st:fn]-4.5,ax_sd[st:fn],linewidth='1',label='X
    axis acceleration IMU');
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[m/s^2]$',fontsize=40)
plt.legend(fontsize=30)
plt.show()
1310

fig2 = plt.figure(figsize=(40, 30), dpi=90)
```

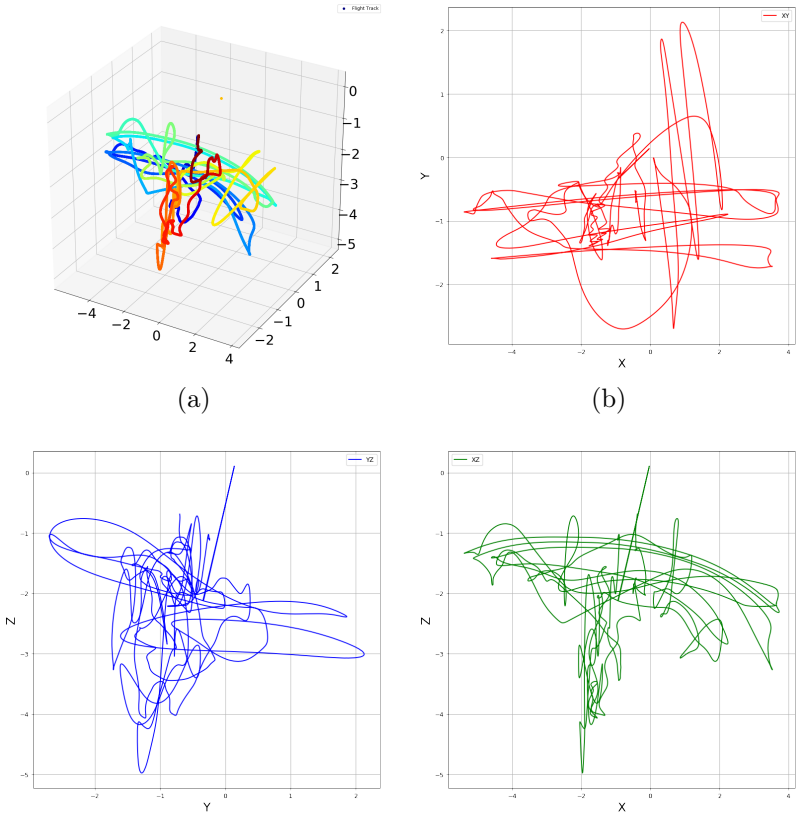
```
plt.grid()
matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)
sf = 1000 # Need to Check this one !!!!!
st = 0*sf; fn = 145*sf;
N = df_opti.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],AY[st:fn],linewidth='1',label='Y axis
    acceleration OptiTrack')
sf = 500;
st = 0*sf; fn = 145*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn]-4.5, ay_sd[st:fn],linewidth='1',label='Y
    axis acceleration IMU');
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[m/s^2]$',fontsize=40)
plt.legend(fontsize=30)
plt.show()

fig3 = plt.figure(figsize=(40, 30), dpi=90)
plt.grid()
matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)
sf = 1000 # Need to Check this one !!!!!
st = 0*sf; fn = 145*sf;
N = df_opti.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],AZ[st:fn],linewidth='1',label='Z axis
    acceleration OptiTrack')
sf = 500;
st = 0*sf; fn = 145*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn]-4.5, az_sd[st:fn],linewidth='1',label='Z
    axis acceleration IMU');
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[m/s^2]$',fontsize=40)
plt.legend(fontsize=30)
```

```
plt.show()
```



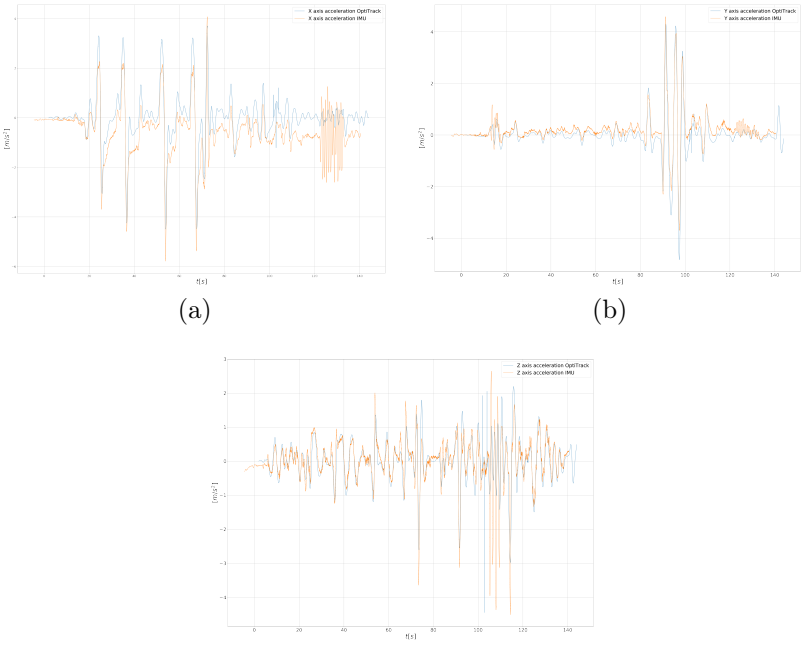
(a) (b) (c) (d) (e) (f) (g)

Figure 62: Filtering comparisons



(a) (b) (c) (d)

Figure 63: OptiTrack flight tracks



(a) (b) (c)

Figure 64: Accelerations comparisons

B Appendix B - Angular Speeds, Accelerations and Moments

```
#!/usr/bin/env python
from __future__ import print_function, division
import numpy as np
import math
import pandas as pd
import re
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import seaborn #plotting lib, but just adding makes the
   matplotlib plots better
import glob # use this for file O later
from numpy import sin, cos, pi, sqrt, dot
import pdb
from mpl_toolkits.mplot3d import Axes3D
import scipy as sp

##FILTRO PASSABASSO

class LowPassFilter(object):

    def __init__(self, alpha):
        self.__setAlpha(alpha)
        self.__y = self.__s = None

    def __setAlpha(self, alpha):
        alpha = float(alpha)
        if alpha<=0 or alpha>1.0:
            raise ValueError("alpha (%s) should be in
   (0.0, 1.0]"%alpha)
        self.__alpha = alpha

    def __call__(self, value, timestamp=None, alpha=None):
        if alpha is not None:
            self.__setAlpha(alpha)
```

```
        if self.__y is None:
            s = value
        else:
            s = self.__alpha*value + (1.0-self.__alpha)*
    self.__s
        self.__y = value
        self.__s = s
        return s

    def lastValue(self):
        return self.__y

##ONEEUROFILTER FOR NOISE FILTERING

class OneEuroFilter(object):

    def __init__(self, freq, mincutoff=1.0, beta=0.0,
    dcutoff=1.0):
        if freq<=0:
            raise ValueError("freq should be >0")
        if mincutoff<=0:
            raise ValueError("mincutoff should be >0")
        if dcutoff<=0:
            raise ValueError("dcutoff should be >0")
        self.__freq = float(freq)
        self.__mincutoff = float(mincutoff)
        self.__beta = float(beta)
        self.__dcutoff = float(dcutoff)
        self.__x = LowPassFilter(self.__alpha(self.
    __mincutoff))
        self.__dx = LowPassFilter(self.__alpha(self.
    __dcutoff))
        self.__lasttime = None

    def __alpha(self, cutoff):
        te    = 1.0 / self.__freq
        tau   = 1.0 / (2*math.pi*cutoff)
        return  1.0 / (1.0 + tau/te)
```

```
    def __call__(self, x, timestamp=None):
        # ---- update the sampling frequency based on timestamps
        if self.__lasttime and timestamp:
            self.__freq = 1.0 / (timestamp-self.__lasttime)
        self.__lasttime = timestamp
        # ---- estimate the current variation per second
        prev_x = self.__x.lastValue()
        dx = 0.0 if prev_x is None else (x-prev_x)*self.__freq # FIXME: 0.0 or value?
        edx = self.__dx(dx, timestamp, alpha=self.__alpha(self.__dcutoff))
        # ---- use it to update the cutoff frequency
        cutoff = self.__mincutoff + self.__beta*math.fabs(edx)
        # ---- filter the given value
        return self.__x(x, timestamp, alpha=self.__alpha(cutoff))

def Euro_filter(signal,config):
    f = OneEuroFilter(**config)
    timestamp = 0.0 # seconds
    filtered = np.zeros([int(len(signal))])
    for i in range(len(signal)):
        filtered[i] = f(signal[i], timestamp)
        timestamp += 1.0/config["freq"]
    return filtered

# READ THE SD CARD

file_name = "pprzlog_0088.LOG"; # Carica il file dati registrati sulla SD a bordo

dt = np.dtype('i4');
di = np.fromfile(file_name, dtype=dt);

dt = np.dtype('f4');
df = np.fromfile(file_name, dtype=dt);
```

```
1100
col = 12+26;
exc = np.mod(di.size,col);
row = int((di.size-exc)/col);

1105 di = di[:-exc].reshape(row,col);
df = df[:-exc].reshape(row,col);

d = np.zeros([row,col]);
d[:,0:12]  = di[:,0:12];
1110 d[:,12:]  = df[:,12:];
data = d.copy()

# Sample frequency
sf = 500;
1115 fo_c = 0.025 #First order actuator dynamics constant (
    discrete, depending on sf)
N = data.shape[0]
t = np.arange(N)/sf
counter = data[:,0]
quat = data[:,15:19]
1120 pos = data[:,19:22]
speed = data[:,22:25]
accel_ned = data[:,32:35]

roll_nf = np.zeros(len(quat))
1125 pitch_nf = np.zeros(len(quat))
yaw_nf = np.zeros(len(quat))

for i in range(0,len(quat)):
    e0 = quat[i,0]
1130    e1 = quat[i,1]
    e2 = quat[i,2]
    e3 = quat[i,3]

    #roll (x-axis rotation)
1135    arg1 = np.arctan2((e2*e3+e0*e1),(+1/2-(e1**2+e2**2)))
    roll_nf[i] = arg1
```

```
    #pitch (y-axis rotation)
    arg2 = (+2.0*(-e1*e3+e0*e2))
    if (math.fabs(arg2) >= 1):
        arg2 = copysign(math.pi/2,arg2); # use 90 degrees if out of range
        pitch_nf[i] = arg2
    else:
        arg2 = np.arcsin(arg2);
        pitch_nf[i] = arg2

    #yaw (z-axis rotation)
    arg3 = np.arctan2((e1*e2+e0*e3),(+1/2-(e2**2+e3**2)))
    yaw_nf[i] = arg3

config = {
    'freq': 500,        # Hz
    'mincutoff': 1,  # 1.0 FIXME
    'beta': 0.1,        # 1.0 FIXME
    'dcutoff': 1.0      # 1.0 this one should be ok
    }
roll = Euro_filter(roll_nf,config)
config = {
    'freq': 500,        # Hz
    'mincutoff': 0.3,  # 1.0 FIXME
    'beta': 0.1,        # 1.0 FIXME
    'dcutoff': 1.0      # 1.0 this one should be ok
    }
pitch = Euro_filter(pitch_nf,config)
config = {
    'freq': 500,        # Hz
    'mincutoff': 0.3,  # 1.0 FIXME
    'beta': 0.1,        # 1.0 FIXME
    'dcutoff': 1.0      # 1.0 this one should be ok
    }
yaw = Euro_filter(yaw_nf,config)

fig = plt.figure(figsize=(40,30), dpi=90)
plt.grid()
matplotlib.rc('xtick', labelsize=30)
```

```
matplotlib.rc('ytick', labelsize=30)
plt.plot(roll_nf,label='Roll angle IMU Not filtered')
plt.plot(roll,label='Roll angle IMU Filtered')
#plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[rad]$',fontsize=40)
plt.legend(fontsize=30);plt.show()
fig = plt.figure(figsize=(40,30), dpi=90)
plt.grid()
plt.plot(pitch_nf,label='Pitch angle IMU Not filtered')
plt.plot(pitch,label='Pitch angle IMU Filtered')
#plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[rad]$',fontsize=40)
plt.legend(fontsize=30);plt.show()
fig = plt.figure(figsize=(40,30), dpi=60)
plt.grid()
plt.plot(yaw_nf,label='Yaw angle IMU Not filtered')
plt.plot(yaw,label='Yaw angle IMU Filtered')
#plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[rad]$',fontsize=40)
plt.legend(fontsize=30);plt.show()

roll = pd.DataFrame(roll)
pitch = pd.DataFrame(pitch)
yaw = pd.DataFrame(yaw)

sf = 500
diff_period = sf
p_sd = roll.diff(periods=diff_period)
q_sd = pitch.diff(periods=diff_period)
r_sd = yaw.diff(periods=diff_period)

config = {
    'freq': 500,        # Hz
    'mincutoff': 0.5,  # 1.0 FIXME
    'beta': 0.01,        # 1.0 FIXME
    'dcutoff': 0.5     # 1.0 this one should be ok
    }
vx_sd = Euro_filter(speed[:,0],config)
config = {
```

```
    'freq': 500,        # Hz
    'mincutoff': 0.5,   # 1.0 FIXME
    'beta': 0.01,        # 1.0 FIXME
    'dcutoff': 0.5      # 1.0 this one should be ok
    }
vy_sd = Euro_filter(speed[:,1],config)
config = {
    'freq': 500,        # Hz
    'mincutoff': 0.5,   # 1.0 FIXME
    'beta': 0.01,        # 1.0 FIXME
    'dcutoff': 0.5      # 1.0 this one should be ok
    }
vz_sd = Euro_filter(speed[:,2],config)

matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)
fig1 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
plt.plot(speed[:,0], linewidth='1',label='Vx IMU Not
   filtered')
plt.plot(vx_sd, linewidth='1',label='Vx IMU Filtered') #
   smooth by filter
#plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[m/s]$',fontsize=40)
plt.legend(fontsize='30')
fig2 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
plt.plot(speed[:,1], linewidth='1',label='Vy IMU Not
   filtered')
plt.plot(vy_sd, linewidth='1',label='Vy IMU Filtered') #
   smooth by filter
#plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[m/s]$',fontsize=40)
plt.legend(fontsize='30')
fig3 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
plt.plot(speed[:,2], linewidth='1',label='Vz IMU Not
   filtered')
```

```
plt.plot(vz_sd, linewidth='1',label='Vz IMU Filtered') # smooth by filter
#plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[m/s]$',fontsize=40)
plt.legend(fontsize='30')

# READ THE OPTITRACK FILE

df_opti = pd.read_csv('DarkO1_15_03_2019_voliere.csv', skiprows=6)
df_opti.drop(['Frame'],axis=1,inplace=True)
sf = 1000 # Need to Check this one !!!!!
st = 0*sf; fn = 145*sf;

Px = df_opti[st:fn].PX
Py = df_opti[st:fn].PY
Pz = df_opti[st:fn].PZ

diff_period = sf
VX = Px.diff(periods=diff_period)
VY = Py.diff(periods=diff_period)
VZ = Pz.diff(periods=diff_period)

W = df_opti[st:fn].W
X = df_opti[st:fn].X
Y = df_opti[st:fn].Y
Z = df_opti[st:fn].Z
Time = df_opti[st:fn].Time
w = W
x = -Z
y = X
z = -Y
t = Time
roll_nf = np.zeros(len(w))
pitch_nf = np.zeros(len(w))
yaw_nf = np.zeros(len(w))
time = np.zeros(len(w))

for i in range(0,len(w)):
```

```
    e0 = w[i]
    e1 = x[i]
    e2 = y[i]
    e3 = z[i]
    time[i] = t[i]

    #roll (x-axis rotation)
    arg1 = np.arctan2((e2*e3+e0*e1),(+1/2-(e1**2+e2**2)))
    roll_nf[i] = arg1

    #pitch (y-axis rotation)
    arg2 = (+2.0*(-e1*e3+e0*e2))
    if (math.fabs(arg2) >= 1):
        arg2 = copysign(math.pi/2,arg2); # use 90 degrees
   if out of range
        pitch_nf[i] = arg2
    else:
        arg2 = np.arcsin(arg2);
        pitch_nf[i] = arg2

    #yaw (z-axis rotation)
    arg3 = np.arctan2((e1*e2+e0*e3),(+1/2-(e2**2+e3**2)))
    yaw_nf[i] = arg3

config = {
    'freq': 1000,        # Hz
    'mincutoff': 0.3,  # 1.0 FIXME
    'beta': 0.1,         # 1.0 FIXME
    'dcutoff': 1.0      # 1.0 this one should be ok
    }
roll = Euro_filter(roll_nf,config)
config = {
    'freq': 1000,        # Hz
    'mincutoff': 0.3,  # 1.0 FIXME
    'beta': 0.1,         # 1.0 FIXME
    'dcutoff': 1.0      # 1.0 this one should be ok
    }
pitch = Euro_filter(pitch_nf,config)
config = {
```

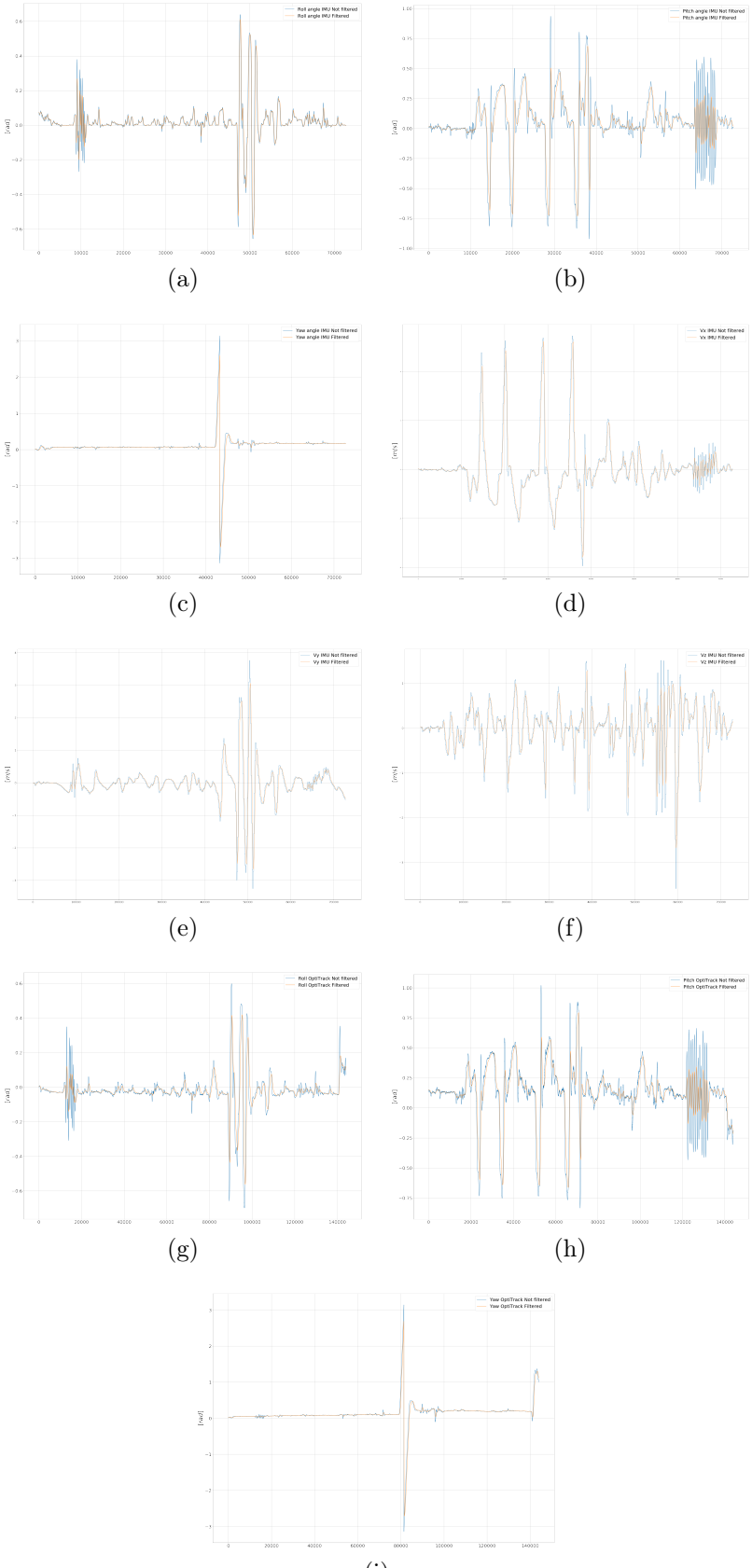
```
    'freq': 1000,        # Hz
    'mincutoff': 0.4,  # 1.0 FIXME
    'beta': 0.1,         # 1.0 FIXME
    'dcutoff': 1.0       # 1.0 this one should be ok
    }
yaw = Euro_filter(yaw_nf,config)

matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)
fig = plt.figure(figsize=(40,30), dpi=90)
plt.grid()
plt.plot(roll_nf,label='Roll OptiTrack Not filtered')
plt.plot(roll,label='Roll OptiTrack Filtered')
#plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[rad]$',fontsize=40)
plt.legend(fontsize=30);plt.show()
fig = plt.figure(figsize=(40,30), dpi=90)
plt.grid()
plt.plot(pitch_nf,label='Pitch OptiTrack Not filtered')
plt.plot(pitch,label='Pitch OptiTrack Filtered')
#plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[rad]$',fontsize=40)
plt.legend(fontsize=30);plt.show()
fig = plt.figure(figsize=(40,30), dpi=60)
plt.grid()
plt.plot(yaw_nf,label='Yaw OptiTrack Not filtered')
plt.plot(yaw,label='Yaw OptiTrack Filtered')
#plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[rad]$',fontsize=40)
plt.legend(fontsize=30);plt.show()

roll = pd.DataFrame(roll)
pitch = pd.DataFrame(pitch)
yaw = pd.DataFrame(yaw)

sf = 1000
diff_period = sf
p_opti = roll.diff(periods=diff_period)
q_opti = pitch.diff(periods=diff_period)
```

```
r_opti = yaw.diff(periods=diff_period)

matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)
fig1 = plt.figure(figsize=(40, 30), dpi=90)
plt.grid()
sf = 1000
st = 0*sf; fn = 145*sf;
N = df_opti.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],p_opti[st:fn],linewidth='1',label='Roll
    rate OptiTrack')
sf = 500;
st =sf; fn = 145*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn]-5.1, p_sd[st:fn],linewidth='1.5',label='
    Roll rate IMU');
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[rad/s]$',fontsize=40)
plt.legend(fontsize=30);plt.show()

fig2 = plt.figure(figsize=(40, 30), dpi=90)
plt.grid()
sf = 1000
st = 0*sf; fn = 145*sf;
N = df_opti.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],q_opti[st:fn],linewidth='1',label='Pitch
    rate OptiTrack')
sf = 500;
st = 0*sf; fn = 145*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn]-5.1, q_sd[st:fn],linewidth='1',label='
    Pitch rate IMU');
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[rad/s]$',fontsize=40)
plt.legend(fontsize=30)
```

```
plt.show()

fig3 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
sf = 1000
st = 0*sf; fn = 145*sf;
N = df_opti.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],r_opti[st:fn],linewidth='1',label='Yaw rate OptiTrack')
sf = 500;
st = 0*sf; fn = 145*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn]-5.1, r_sd[st:fn],linewidth='1',label='Yaw rate IMU');
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[rad/s]$',fontsize=40)
plt.legend(fontsize=30)
plt.show()

sf = 1000
diff_period = sf
pdot_opti = p_opti.diff(periods=diff_period)
qdot_opti = q_opti.diff(periods=diff_period)
rdot_opti = r_opti.diff(periods=diff_period)

sf = 500
diff_period = sf
pdot_sd = p_sd.diff(periods=diff_period)
qdot_sd = q_sd.diff(periods=diff_period)
rdot_sd = r_sd.diff(periods=diff_period)

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

matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)
fig1 = plt.figure(figsize=(40, 30), dpi=90)
plt.grid()
```

```
sf = 1000
st = 0*sf; fn = 145*sf;
N = df_opti.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],pdot_opti[st:fn],linewidth='1',label='
    Roll second derivative OptiTrack')
sf = 500;
st =0*sf; fn = 145*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn]-5.1, pdot_sd[st:fn],linewidth='1.5',
    label='Roll second derivative IMU');
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[rad/s^2]$',fontsize=40)
plt.legend(fontsize=30);plt.show()

fig2 = plt.figure(figsize=(40, 30), dpi=90)
plt.grid()
sf = 1000
st = 0*sf; fn = 145*sf;
N = df_opti.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],qdot_opti[st:fn],linewidth='1',label='
    Pitch second derivative OptiTrack')
sf = 500;
st = 0*sf; fn = 145*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn]-5.1, qdot_sd[st:fn],linewidth='1',label=
    'Pitch second derivative IMU');
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[rad/s^2]$',fontsize=40)
plt.legend(fontsize=30)
plt.show()

fig3 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
sf = 1000
st = 0*sf; fn = 145*sf;
```

```
N = df_opti.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],rdot_opti[st:fn],linewidth='1',label='
    Yaw second derivative OptiTrack')
sf = 500;
st = 0*sf; fn = 145*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn]-5.1, rdot_sd[st:fn],linewidth='1',label=
    'Yaw second derivative IMU');
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[rad/s^2]$',fontsize=40)
plt.legend(fontsize=30)
plt.show()
```



(a) (b) (c) (d) (e) (f) (g) (h) (i)

Figure 65: Filtering comparisons

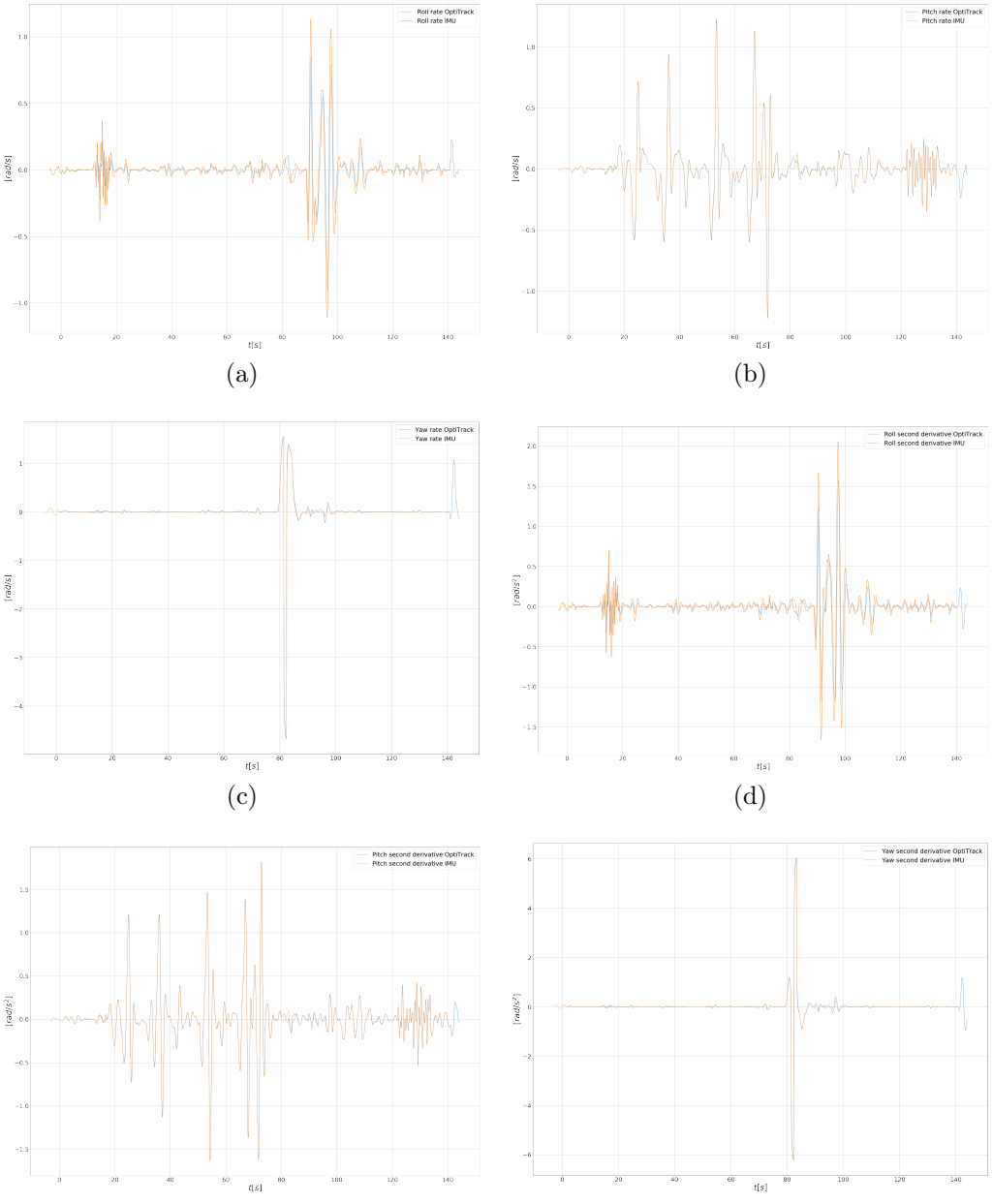


Figure 66: Angular speeds and accelerations comparisons

C Appendix C - Indoor flight FFT

```
#!/usr/bin/env python
from __future__ import print_function, division
import numpy as np
import math
import pandas as pd
import re
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import seaborn #plotting lib, but just adding makes the
    matplotlib plots better
import glob # use this for file O later
from numpy import sin, cos, pi, sqrt, dot
from scipy import stats, optimize
from scipy import linalg as la
from scipy.optimize import leastsq
from scipy.optimize import least_squares
import pdb
from mpl_toolkits.mplot3d import Axes3D
from scipy.interpolate import griddata, interp1d
from scipy import signal
from scipy.optimize import curve_fit
from scipy import interpolate
import scipy as sp

# Read the binary log

file_name = "pprzlog_0088.LOG"; # Carica il file dati
    registrati sulla SD a bordo

dt = np.dtype('i4');
di = np.fromfile(file_name, dtype=dt);

dt = np.dtype('f4');
df = np.fromfile(file_name, dtype=dt);
```

```
col = 12+26;
exc = np.mod(di.size,col);
row = int((di.size-exc)/col);

di = di[:-exc].reshape(row,col);
df = df[:-exc].reshape(row,col);

d = np.zeros([row,col]);
d[:,0:12]  = di[:,0:12];
d[:,12:]   = df[:,12:];
data = d.copy()

# Sample frequency
sf = 500;
fo_c = 0.025 #First order actuator dynamics constant (
   discrete, depending on sf)
N = data.shape[0]
t = np.arange(N)/sf
counter = data[:,0]
airspeed = data[:,11]/2**19
quat = data[:,15:19]
pos = data[:,19:22]
speed = data[:,22:25]
accel_ned = data[:,32:35]

roll_sd_nf = np.zeros(len(quat))
pitch_sd_nf = np.zeros(len(quat))
yaw_sd_nf = np.zeros(len(quat))

for i in range(0,len(quat)):
    e0 = quat[i,0]
    e1 = quat[i,1]
    e2 = quat[i,2]
    e3 = quat[i,3]

    #roll (x-axis rotation)
    arg1 = np.arctan2((e2*e3+e0*e1),(+1/2-(e1**2+e2**2)))
    roll_sd_nf[i] = arg1
```

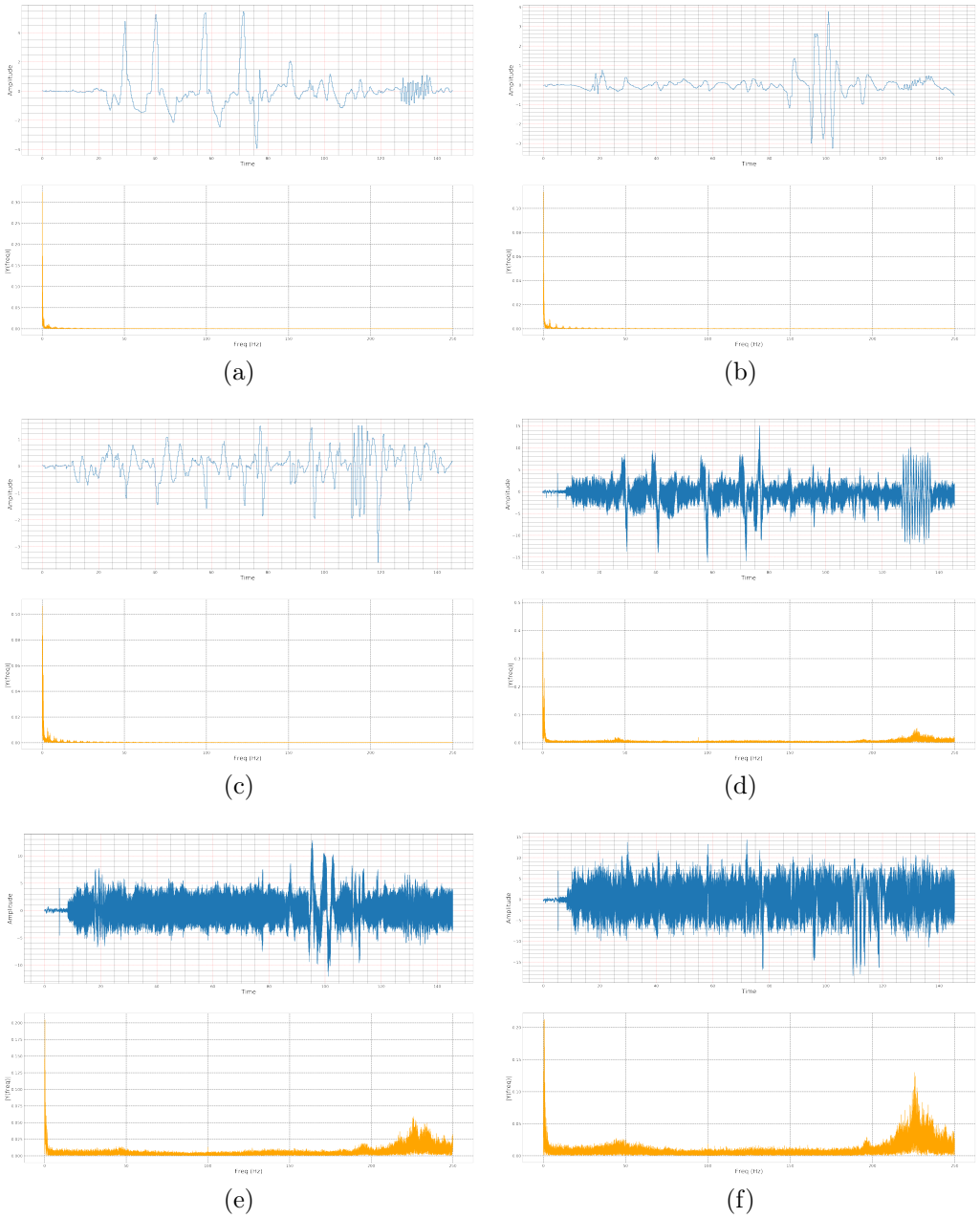
```
    #pitch (y-axis rotation)
    arg2 = (+2.0*(-e1*e3+e0*e2))
    if (math.fabs(arg2) >= 1):
        arg2 = copysign(math.pi/2,arg2); # use 90 degrees if out of range
        pitch_sd_nf[i] = arg2
    else:
        arg2 = np.arcsin(arg2);
        pitch_sd_nf[i] = arg2

    #yaw (z-axis rotation)
    arg3 = np.arctan2((e1*e2+e0*e3),(+1/2-(e2**2+e3**2)))
    yaw_sd_nf[i] = arg3

df_opti = pd.read_csv('DarkO1_15_03_2019_voliere.csv', skiprows=6)
df_opti.drop(['Frame'],axis=1,inplace=True)
sf = 1000 # Need to Check this one !!!!!
st = 0*sf; fn = 145*sf;
W = df_opti[st:fn].W
X = df_opti[st:fn].X
Y = df_opti[st:fn].Y
Z = df_opti[st:fn].Z
Time = df_opti[st:fn].Time
w = W
x = X
y = Y
z = Z
time = Time
roll_opti_nf = np.zeros(len(w))
pitch_opti_nf = np.zeros(len(w))
yaw_opti_nf = np.zeros(len(w))
t = np.zeros(len(w))

for i in range(0,len(w)):
    e0 = w[i]
    e1 = x[i]
    e2 = y[i]
    e3 = z[i]
```

```
    t[i] = time[i]

    #roll (x-axis rotation)
    arg1 = np.arctan2((e2*e3+e0*e1),(+1/2-(e1**2+e2**2)))
    roll_opti_nf[i] = arg1

    #pitch (y-axis rotation)
    arg2 = (+2.0*(-e1*e3+e0*e2))
    if (math.fabs(arg2) >= 1):
        arg2 = copysign(math.pi/2,arg2); # use 90 degrees
   if out of range
        pitch_opti_nf[i] = arg2
    else:
        arg2 = np.arcsin(arg2);
        pitch_opti_nf[i] = arg2

    #yaw (z-axis rotation)
    arg3 = np.arctan2((e1*e2+e0*e3),(+1/2-(e2**2+e3**2)))
    yaw_opti_nf[i] = arg3

def plot_fft(signal,freq):

    Fs = freq;  # sampling rate
    y = signal;
    n = len(y) # length of the signal
    duration = n/Fs; # signal duration
    Ts = 1.0/Fs; # sampling interval
    t = np.arange(0,duration,Ts) # time vector

    # ff = 10;   # frequency of the signal
    # y = np.sin(2*np.pi*ff*t) + np.sin(2*np.pi*ff*t*10)

    k = np.arange(n)
    T = n/Fs
    frq = k/T # two sides frequency range
    frq = frq[range(int(n/2))] # one side frequency range

    Y = np.fft.fft(y)/n # fft computing and normalization
    Y = Y[range(int(n/2))]
```

```
    fig, ax = plt.subplots(2, 1,figsize=(40,30),dpi=90)
    matplotlib.rc('xtick', labelsize=20)
    matplotlib.rc('ytick', labelsize=20)
    ax[0].plot(t,y)
    ax[0].set_xlabel('Time',fontsize=30)
    ax[0].set_ylabel('Amplitude',fontsize=30)
    ax[1].plot(frq,abs(Y),'orange') # plotting the spectrum
    ax[1].set_xlabel('Freq (Hz)',fontsize=30)
    ax[1].set_ylabel('|Y(freq)|',fontsize=30)

    # Turn on the minor TICKS, which are required for the minor GRID
    ax[0].minorticks_on()
    # Customize the major grid
    ax[0].grid(which='major', linestyle='--', linewidth='0.5', color='red')
    # Customize the minor grid
    ax[0].grid(which='minor', linestyle='-', linewidth='0.5', color='black')
    plt.grid(True, lw = 2, ls = '--', c = '.5');plt.show()

print('roll_sd_nf')
plot_fft(roll_sd_nf,500)
print('pitch_sd_nf')
plot_fft(pitch_sd_nf,500)
print('yaw_sd_nf')
plot_fft(yaw_sd_nf,500)

print('roll_opti_nf')
plot_fft(roll_opti_nf,1000)
print('pitch_opti_nf')
plot_fft(pitch_opti_nf,1000)
print('yaw_opti_nf')
plot_fft(yaw_opti_nf,1000)

print('speed[:,0]')
plot_fft(speed[:,0],500)
```

```
print('speed[:,1]')
plot_fft(speed[:,1],500)
print('speed[:,2]')
plot_fft(speed[:,2],500)

print('airspeed')
plot_fft(airspeed,500)


print('accel_ned[:,0]')
plot_fft(accel_ned[:,0],500)
print('accel_ned[:,1]')
plot_fft(accel_ned[:,1],500)
print('accel_ned[:,2]')
plot_fft(accel_ned[:,2],500)
```

Figure 67: FFT Euler angles output



(a) (b) (c) (d) (e) (f)

Figure 68: FFT speeds and accelerations output

D Appendix D - Flight Analysis

```
#!/usr/bin/env python

from __future__ import print_function, division
import numpy as np
import pandas as pd
import math
import re
import matplotlib
import matplotlib.pyplot as plt
import seaborn #plotting lib, but just adding makes the
    matplotlob plots better
import glob # use this for file IO later

from numpy import sin, cos, pi, sqrt, dot
import scipy as sp
from scipy import stats, optimize
from scipy import linalg as la
from scipy.optimize import leastsq
from scipy.optimize import least_squares
import pdb

import matplotlib.mlab as mlab
import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D
from scipy.interpolate import griddata, interp1d
from scipy import signal
from scipy.optimize import curve_fit
from scipy import interpolate

# Read the binary log
file_name = "pprzlog_0099.LOG" #Long flight with different
    airspeeds

dt = np.dtype('i4');
di = np.fromfile(file_name, dtype=dt);
```

```
dt = np.dtype('f4');
df = np.fromfile(file_name, dtype=dt);

col = 12+26 ;
exc = np.mod(di.size,col) ;
row = int((di.size-exc)/col);

di = di[:-exc].reshape(row,col);
df = df[:-exc].reshape(row,col);

d = np.zeros([row,col]);
d[:,0:12]  = di[:,0:12];
d[:,12:]   = df[:,12:];
data = d.copy()

# Sample frequency
sf = 512;
#First order actuator dynamics constant (discrete,
   depending on sf)
fo_c = 0.025

N = data.shape[0]

# Data structure
t = np.arange(N)/sf
counter  = data[:,0]
act      = data[:,1:5] #/9600 # roll-pitch-yaw
quatsp   = data[:,5:9]/2**15 # quat i-x-y-z
airspeed = data[:,11]/2**19

gyro = data[:,12:15] # roll-pitch-yaw
quat = data[:,15:19] # quat i-x-y-z
pos  = data[:,19:22]
speed= data[:,22:25] # NED speed ?
indi_v = data[:,25:29]
accel_sp  = data[:,29:32]
accel_ned = data[:,32:35]
speed_sp  = data[:,35:38]
```

```
# plt.subplot(427);
sf = 500
st = 0*sf; fn = N;
# st = 0*sf; fn = 320*sf;
fig = plt.figure(figsize=(15,15), dpi=90)
ax = plt.axes(projection='3d')
#ax = fig.add_subplot(4,2,7, projection='3d');
# ax.plot3D(pos[st:fn,0],pos[st:fn,1],-pos[st:fn,2], "o",
    label='IMU Track');
cm = plt.cm.jet(np.linspace(0,1,(fn-st)))
ax.scatter(pos[st:fn,0],pos[st:fn,1],-pos[st:fn,2], "-", c
   =cm, label='Flight Track');
plt.legend();
plt.show()
fig = plt.figure(figsize=(10,10), dpi=90)
matplotlib.rc('xtick', labelsize=10)
matplotlib.rc('ytick', labelsize=10)
plt.grid()
plt.plot(pos[:,0],pos[:,1],color = 'r',label = 'XY')
plt.xlabel('X',fontsize=15)
plt.ylabel('Y',fontsize=15)
plt.legend();
fig = plt.figure(figsize=(10,10), dpi=90)
plt.grid()
plt.plot(pos[:,1],pos[:,2],color = 'b',label = 'YZ')
plt.xlabel('Y',fontsize=15)
plt.ylabel('Z',fontsize=15)
plt.legend();
fig = plt.figure(figsize=(10,10), dpi=90)
plt.grid()
plt.plot(pos[:,0],pos[:,2],color = 'g',label = 'XZ')
plt.xlabel('X',fontsize=15)
plt.ylabel('Z',fontsize=15)
plt.legend();

#Isolate flight phases

#0-360 inner & outer circles COUNTERCLOCKWISE
```

```
#150-250 inner circles COUNTERCLOCKWISE
#600-700 inner circles CLOCKWISE
#360-550 outer circles CLOCKWISE
#500-750 passage from outer to inner
#360-400 one circle
#90-750 circles (horizontal flight)
#90-1050 actual flight

fig = plt.figure(figsize=(10,10), dpi=90)
matplotlib.rc('xtick', labelsize=10)
matplotlib.rc('ytick', labelsize=10)
plt.grid()
plt.plot(pos[150*500:250*500,0],pos[150*500:250*500,1],
   color = 'b',label = 'XY')
plt.plot(pos[600*500:700*500,0],pos[600*500:700*500,1],
   color = 'r')
plt.xlabel('X',fontsize=15)
plt.ylabel('Y',fontsize=15)
plt.legend();
fig = plt.figure(figsize=(10,10), dpi=90)
plt.grid()
plt.plot(pos[70*500:1050*500,1],pos[70*500:1050*500,2],
   color = 'b',label = 'YZ')
plt.legend();
fig = plt.figure(figsize=(10,10), dpi=90)
plt.grid()
plt.plot(pos[0*500:1100*500,0],pos[0*500:1100*500,2],color
    = 'g',label = 'XZ')
plt.legend();

##LOWPASS FILTER

class LowPassFilter(object):

    def __init__(self, alpha):
        self.__setAlpha(alpha)
        self.__y = self.__s = None

    def __setAlpha(self, alpha):
```

```
        alpha = float(alpha)
        if alpha<=0 or alpha>1.0:
            raise ValueError("alpha (%s) should be in
    (0.0, 1.0]"%alpha)
        self.__alpha = alpha

    def __call__(self, value, timestamp=None, alpha=None):
        if alpha is not None:
            self.__setAlpha(alpha)
        if self.__y is None:
            s = value
        else:
            s = self.__alpha*value + (1.0-self.__alpha)*
    self.__s
        self.__y = value
        self.__s = s
        return s

    def lastValue(self):
        return self.__y

##ONEEUROFILTER FOR NOISE FILTERING

class OneEuroFilter(object):

    def __init__(self, freq, mincutoff=1.0, beta=0.0,
    dcutoff=1.0):
        if freq<=0:
            raise ValueError("freq should be >0")
        if mincutoff<=0:
            raise ValueError("mincutoff should be >0")
        if dcutoff<=0:
            raise ValueError("dcutoff should be >0")
        self.__freq = float(freq)
        self.__mincutoff = float(mincutoff)
        self.__beta = float(beta)
        self.__dcutoff = float(dcutoff)
        self.__x = LowPassFilter(self.__alpha(self.
    __mincutoff))
```

```
        self.__dx = LowPassFilter(self.__alpha(self.__dcutoff))
        self.__lasttime = None

    def __alpha(self, cutoff):
        te    = 1.0 / self.__freq
        tau   = 1.0 / (2*math.pi*cutoff)
        return  1.0 / (1.0 + tau/te)

    def __call__(self, x, timestamp=None):
        # ---- update the sampling frequency based on timestamps
        if self.__lasttime and timestamp:
            self.__freq = 1.0 / (timestamp-self.__lasttime)
        self.__lasttime = timestamp
        # ---- estimate the current variation per second
        prev_x = self.__x.lastValue()
        dx = 0.0 if prev_x is None else (x-prev_x)*self.__freq # FIXME: 0.0 or value?
        edx = self.__dx(dx, timestamp, alpha=self.__alpha(self.__dcutoff))
        # ---- use it to update the cutoff frequency
        cutoff = self.__mincutoff + self.__beta*math.fabs(edx)
        # ---- filter the given value
        return self.__x(x, timestamp, alpha=self.__alpha(cutoff))

def Euro_filter(signal,config):
    f = OneEuroFilter(**config)
    timestamp = 0.0 # seconds
    filtered = np.zeros([int(len(signal))])
    for i in range(len(signal)):
        filtered[i] = f(signal[i], timestamp)
        timestamp += 1.0/config["freq"]
    return filtered

roll_nf = np.zeros(len(quat))
```

```
pitch_nf = np.zeros(len(quat))
yaw_nf = np.zeros(len(quat))

for i in range(0,len(quat)):
    e0 = quat[i,0]
    e1 = quat[i,1]
    e2 = quat[i,2]
    e3 = quat[i,3]

    #roll (x-axis rotation)
    arg1 = np.arctan2((e2*e3+e0*e1),(+1/2-(e1**2+e2**2)))
    roll_nf[i] = arg1

    #pitch (y-axis rotation)
    arg2 = (+2.0*(-e1*e3+e0*e2))
    if (math.fabs(arg2) >= 1):
        arg2 = copysign(math.pi/2,arg2); # use 90 degrees
   if out of range
        pitch_nf[i] = arg2
    else:
        arg2 = np.arcsin(arg2);
        pitch_nf[i] = arg2

    #yaw (z-axis rotation)
    arg3 = np.arctan2((e1*e2+e0*e3),(+1/2-(e2**2+e3**2)))
    yaw_nf[i] = arg3

config = {
    'freq': 500,        # Hz
    'mincutoff': 0.15,  # 1.0 FIXME
    'beta': 0.1,        # 1.0 FIXME
    'dcutoff': 1.0      # 1.0 this one should be ok
    }
roll = Euro_filter(roll_nf,config)
config = {
    'freq': 500,        # Hz
    'mincutoff': 0.1,   # 1.0 FIXME
    'beta': 0.1,        # 1.0 FIXME
    'dcutoff': 1.0      # 1.0 this one should be ok
```

```
    }
pitch = Euro_filter(pitch_nf,config)
config = {
    'freq': 500,        # Hz
    'mincutoff': 0.3,   # 1.0 FIXME
    'beta': 0.1,        # 1.0 FIXME
    'dcutoff': 1.0      # 1.0 this one should be ok
    }
yaw = Euro_filter(yaw_nf,config)

#INNER circles negative roll
#OUTER circles negative roll
#BEFORE positive roll

fig = plt.figure(figsize=(40,30), dpi=90)
matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)
plt.grid()
plt.plot(roll_nf,label='Roll angle Not filtered',linewidth
   ='1.5')
plt.plot(roll,label='Roll angle Filtered',linewidth='0.75'
   ,color='red')
#plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[rad]$',fontsize=40)
plt.legend(fontsize=30);plt.show()
fig = plt.figure(figsize=(40,30), dpi=90)
plt.grid()
plt.plot(pitch_nf,label='Pitch angle Not filtered',
   linewidth='1.5')
plt.plot(pitch,label='Pitch angle Filtered',linewidth='
   0.75',color='red')
#plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[rad]$',fontsize=40)
plt.legend(fontsize=30);plt.show()
fig = plt.figure(figsize=(40,30), dpi=60)
plt.grid()
plt.plot(yaw_nf,label='Yaw angle Not filtered',linewidth='
   1.5')
```

```
plt.plot(yaw,label='Yaw angle Filtered',linewidth='0.75',
    color='red')
#plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[rad]$',fontsize=40)
plt.legend(fontsize=30);plt.show()

pitch1 = pitch
roll = pd.DataFrame(roll)
pitch = pd.DataFrame(pitch)
yaw = pd.DataFrame(yaw)

sf = 500
diff_period = sf
p_sd = roll.diff(periods=diff_period)
q_sd = pitch.diff(periods=diff_period)
r_sd = yaw.diff(periods=diff_period)

fig = plt.figure(figsize=(40,30), dpi=90)
matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)
plt.grid()
sf = 500;
st = 150*sf; fn = 250*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],p_sd[st:fn],label='Rolling speed',
    linewidth='0.75',color='red')
st = 600*sf; fn = 700*sf;
plt.plot(t[st:fn],p_sd[st:fn],linewidth='0.75',color='red'
    )
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[rad]$',fontsize=40)
plt.legend(fontsize=30);plt.show()
fig = plt.figure(figsize=(40,30), dpi=90)
matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)
plt.grid()
sf = 500;
st = 150*sf; fn = 250*sf;
```

```
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],q_sd[st:fn],label='Pitching speed',
    linewidth='0.75',color='red')
st = 600*sf; fn = 700*sf;
plt.plot(t[st:fn],q_sd[st:fn],label='Pitching speed',
    linewidth='0.75',color='red')
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[rad]$',fontsize=40)
plt.legend(fontsize=30);plt.show()
fig = plt.figure(figsize=(40,30), dpi=90)
matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)
plt.grid()
sf = 500;
st = 150*sf; fn = 250*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],r_sd[st:fn],label='Yawing speed',
    linewidth='0.75',color='red')
st = 600*sf; fn = 700*sf;
plt.plot(t[st:fn],r_sd[st:fn],linewidth='0.75',color='red'
    )
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[rad]$',fontsize=40)
plt.legend(fontsize=30);plt.show()

##CONFIGURATION AND USE OF ONEEUROFILTER

config = {
     'freq': 500,        # Hz
     'mincutoff': 0.1,   # 1.0 FIXME
     'beta': 0.01,        # 1.0 FIXME
     'dcutoff': 0.5      # 1.0 this one should be ok
     }
v_sd = Euro_filter(airspeed,config)
config = {
     'freq': 500,        # Hz
     'mincutoff': 0.2,   # 1.0 FIXME
```

```
    'beta': 0.01,        # 1.0 FIXME
    'dcutoff': 0.5      # 1.0 this one should be ok
    }
vx_sd = Euro_filter(speed[:,0],config)
config = {
    'freq': 500,        # Hz
    'mincutoff': 0.2,   # 1.0 FIXME
    'beta': 0.01,        # 1.0 FIXME
    'dcutoff': 0.5      # 1.0 this one should be ok
    }
vy_sd = Euro_filter(speed[:,1],config)
config = {
    'freq': 500,        # Hz
    'mincutoff': 0.3,   # 1.0 FIXME
    'beta': 0.01,        # 1.0 FIXME
    'dcutoff': 0.5      # 1.0 this one should be ok
    }
vz_sd = Euro_filter(speed[:,2],config)
config = {
    'freq': 500,        # Hz
    'mincutoff': 0.1,   # 1.0 FIXME
    'beta': 0.01,        # 1.0 FIXME
    'dcutoff': 0.5      # 1.0 this one should be ok
    }
ax_sd = Euro_filter(accel_ned[:,0],config)
config = {
    'freq': 500,        # Hz
    'mincutoff': 0.1,   # 1.0 FIXME
    'beta': 0.01,        # 1.0 FIXME
    'dcutoff': 0.5      # 1.0 this one should be ok
    }
ay_sd = Euro_filter(accel_ned[:,1],config)
config = {
    'freq': 500,        # Hz
    'mincutoff': 0.2,   # 1.0 FIXME
    'beta': 0.0001,        # 1.0 FIXME
    'dcutoff': 0.05      # 1.0 this one should be ok
    }
az_sd = Euro_filter(accel_ned[:,2],config)
```

```
fig3 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
plt.plot(airspeed, linewidth='1', linestyle="-", c='r',
    label='Airspeed Not filtered')
plt.plot(v_sd, linewidth='1', linestyle="-", c="b",label='
    Airspeed Filtered') # smooth by filter
plt.ylabel('$[m/s]$',fontsize=40)
plt.legend(fontsize='30')
fig1 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
matplotlib.rc('xtick', labelsize=20)
matplotlib.rc('ytick', labelsize=20)
plt.plot(speed[:,0], linewidth='1', c='r',label='VX Not
    filtered')
plt.plot(vx_sd, linewidth='1', c="b",label='VX Filtered')
     # smooth by filter
plt.ylabel('$[m/s]$',fontsize=40)
plt.legend(fontsize='30')
fig2 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
plt.plot(speed[:,1], linewidth='1', linestyle="-", c='r',
    label='VY Not filtered')
plt.plot(vy_sd, linewidth='1', linestyle="-", c="b",label=
    'VY Filtered') # smooth by filter
plt.ylabel('$[m/s]$',fontsize=40)
plt.legend(fontsize='30')
fig3 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
plt.plot(speed[:,2], linewidth='1', linestyle="-", c='r',
    label='VZ Not filtered')
plt.plot(vz_sd, linewidth='1', linestyle="-", c="b",label=
    'VZ Filtered') # smooth by filter
plt.ylabel('$[m/s]$',fontsize=40)
plt.legend(fontsize='30')
fig4 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
plt.plot(accel_ned[:,0], linewidth='0.1', linestyle="-", c
    ='r',label='AX Not filtered')
```

```
plt.plot(ax_sd, linewidth='1', linestyle="-", c="b",label=
    'AX Filtered') # smooth by filter
plt.ylabel('$[m/s^2]$',fontsize=40)
plt.legend(fontsize='30')
fig4 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
plt.plot(accel_ned[:,1], linewidth='0.1', linestyle="-", c
    ='r',label='AY Not filtered')
plt.plot(ay_sd, linewidth='1', linestyle="-", c="b",label=
    'AY Filtered') # smooth by filter
plt.ylabel('$[m/s^2]$',fontsize=40)
plt.legend(fontsize='30')
fig4 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
plt.plot(accel_ned[:,2], linewidth='0.1', linestyle="-", c
    ='r',label='AZ Not filtered')
plt.plot(az_sd, linewidth='1', linestyle="-", c="b",label=
    'AZ Filtered') # smooth by filter
plt.ylabel('$[m/s^2]$',fontsize=40)
plt.legend(fontsize='30')

fig3 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
sf = 500;
st = 0*sf; fn = 1100*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],v_sd[st:fn], linewidth='1', linestyle="-
    ", c="b",label='Airspeed Filtered') # smooth by filter
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[m/s]$',fontsize=40)
plt.legend(fontsize='30')
fig1 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
sf = 500;
st = 0*sf; fn = 1100*sf;
N = data.shape[0]
t = np.arange(N)/sf
matplotlib.rc('xtick', labelsize=20)
```

```
matplotlib.rc('ytick', labelsize=20)
plt.plot(t[st:fn],vx_sd[st:fn], linewidth='1', c="b",label
    ='VX Filtered')  # smooth by filter
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[m/s]$',fontsize=40)
plt.legend(fontsize='30')
fig2 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
sf = 500;
st = 0*sf; fn = 1100*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],vy_sd[st:fn], linewidth='1', linestyle="
    -", c="b",label='VY Filtered')  # smooth by filter
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[m/s]$',fontsize=40)
plt.legend(fontsize='30')
fig3 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
sf = 500;
st = 0*sf; fn = 1100*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],vz_sd[st:fn], linewidth='1', linestyle="
    -", c="b",label='VZ Filtered')  # smooth by filter
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[m/s]$',fontsize=40)
plt.legend(fontsize='30')
fig4 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
sf = 500;
st = 0*sf; fn = 1100*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],ax_sd[st:fn], linewidth='1', linestyle="
    -", c="b",label='AX Filtered')  # smooth by filter
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[m/s^2]$',fontsize=40)
plt.legend(fontsize='30')
```

```
fig4 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
sf = 500;
st = 0*sf; fn = 1100*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],ay_sd[st:fn], linewidth='1', linestyle="
    -", c="b",label='AY Filtered')  # smooth by filter
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[m/s^2]$',fontsize=40)
plt.legend(fontsize='30')
fig4 = plt.figure(figsize=(40, 30), dpi=60)
plt.grid()
sf = 500;
st = 0*sf; fn = 1100*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],az_sd[st:fn], linewidth='1', linestyle="
    -", c="b",label='AZ Filtered')  # smooth by filter
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[m/s^2]$',fontsize=40)
plt.legend(fontsize='30')

#Comparison groundspeed and airspeed

v_ground = (vx_sd**2+vy_sd**2+vz_sd**2)**0.5

fig1 = plt.figure(figsize=(40, 30), dpi=90)
plt.grid()
matplotlib.rc('xtick', labelsize=50)
matplotlib.rc('ytick', labelsize=50)
sf = 500;
st = 0*sf; fn = 1100*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],v_ground[st:fn],linewidth='1',color = 'b
    ',label='Ground speed');
plt.plot(t[st:fn],v_sd[st:fn],linewidth='1',color = 'r',
    label='Airspeed');
```

```
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[m/s]$',fontsize=40)
plt.legend(fontsize=30)
plt.show()

alpha_nf = np.zeros(len(v_ground))
#beta_nf = np.zeros(len(v_ground))

for i in range(0,len(v_ground)):
    gamma = np.arctan(vz_sd[i]/vx_sd[i])
    alpha_nf[i] = pitch1[i]-gamma
    #beta_nf[i] = np.arctan(vy_sd[i]/vx_sd[i])

def plot_fft(signal,freq):

    Fs = freq;  # sampling rate
    y = signal;
    n = len(y) # length of the signal
    duration = n/Fs; # signal duration
    Ts = 1.0/Fs; # sampling interval
    t = np.arange(0,duration,Ts) # time vector

    # ff = 10;   # frequency of the signal
    # y = np.sin(2*np.pi*ff*t) + np.sin(2*np.pi*ff*t*10)

    k = np.arange(n)
    T = n/Fs
    frq = k/T # two sides frequency range
    frq = frq[range(int(n/2))] # one side frequency range

    Y = np.fft.fft(y)/n # fft computing and normalization
    Y = Y[range(int(n/2))]

    fig, ax = plt.subplots(2, 1,figsize=(40,30),dpi=90)
    matplotlib.rc('xtick', labelsize=10)
    matplotlib.rc('ytick', labelsize=10)
    ax[0].plot(t,y)
    ax[0].set_xlabel('Time',fontsize=30)
    ax[0].set_ylabel('Amplitude',fontsize=30)
```

```
    ax[1].plot(frq,abs(Y),'blue') # plotting the spectrum
    ax[1].set_xlabel('Freq (Hz)',fontsize=30)
    ax[1].set_ylabel('|Y(freq)|',fontsize=30)

    # Turn on the minor TICKS, which are required for the minor GRID
    ax[1].minorticks_on()
    # Customize the major grid
    ax[1].grid(which='major', linestyle='--', linewidth='0.5', color='red')
    # Customize the minor grid
    ax[1].grid(which='minor', linestyle='-', linewidth='0.5', color='black')
    plt.grid(True, lw = 2, ls = '--', c = '.5');plt.show()

print('alpha_nf')
plot_fft(alpha_nf,500)
#print('beta_nf')
#plot_fft(beta_nf,500)

config = {
    'freq': 500,        # Hz
    'mincutoff': 0.1,   # 1.0 FIXME
    'beta': 0.01,        # 1.0 FIXME
    'dcutoff': 0.5      # 1.0 this one should be ok
    }
alpha = Euro_filter(alpha_nf,config)
#config = {
    #'freq': 500,        # Hz
    #'mincutoff': 0.2,   # 1.0 FIXME
    #'beta': 0.0001,        # 1.0 FIXME
    #'dcutoff': 0.05      # 1.0 this one should be ok
    #}
#beta = Euro_filter(beta_nf,config)

fig1 = plt.figure(figsize=(40, 30), dpi=90)
plt.grid()
matplotlib.rc('xtick', labelsize=20)
matplotlib.rc('ytick', labelsize=20)
```

```
sf = 500;
st = 0*sf; fn = 1100*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],alpha_nf[st:fn],linewidth='1',color = 'b
    ',label='Angle of attack \alpha Not Filtered');
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[rad]$',fontsize=40)
plt.legend(fontsize=30)
plt.show()
#fig1 = plt.figure(figsize=(40, 30), dpi=90)
#plt.grid()
#matplotlib.rc('xtick', labelsize=50)
#matplotlib.rc('ytick', labelsize=50)
#sf = 500;
#st = 0*sf; fn = 1100*sf;
#N = data.shape[0]
#t = np.arange(N)/sf
#plt.plot(t[st:fn],beta_nf[st:fn],linewidth='1.5',color =
    'b',label='Sideslip angle \beta Not Filtered');
#plt.xlabel('$t[s]$',fontsize=40)
#plt.ylabel('$[deg]$',fontsize=40)
#plt.legend(fontsize=30)
#plt.show()
fig1 = plt.figure(figsize=(40, 30), dpi=90)
plt.grid()
matplotlib.rc('xtick', labelsize=20)
matplotlib.rc('ytick', labelsize=20)
sf = 500;
st = 0*sf; fn = 1100*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],alpha[st:fn],linewidth='1',color = 'r',
    label='Angle of attack     Filtered');
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[rad]$',fontsize=40)
plt.legend(fontsize=30)
plt.show()
#fig1 = plt.figure(figsize=(40, 30), dpi=90)
```

```
#plt.grid()
#matplotlib.rc('xtick', labelsize=50)
#matplotlib.rc('ytick', labelsize=50)
#sf = 500;
#st = 0*sf; fn = 1100*sf;
#N = data.shape[0]
#t = np.arange(N)/sf
#plt.plot(t[st:fn],beta[st:fn],linewidth='1',color = 'r',
    label='Sideslip angle    ');
#plt.xlabel('$t[s]$',fontsize=40)
#plt.ylabel('$[deg]$',fontsize=40)
#plt.legend(fontsize=30)
#plt.show()

#FORCES

mass = 0.492
rho = 1.225
S_wet = 0.0743*2

col = {'a_x': ax_sd,'a_y': ay_sd,'a_z': az_sd}
a_sd = pd.DataFrame(col)

FX_sd = mass*a_sd.a_x;
FY_sd = mass*a_sd.a_y;
FZ_sd = mass*(a_sd.a_z+9.81);

L = -FZ_sd*np.cos(alpha)+FX_sd*np.sin(alpha)
D = -FZ_sd*np.sin(alpha)-FX_sd*np.cos(alpha)

CD = 2*D/(rho*v_sd**2*S_wet)
CY_sd = 2*FY_sd/(rho*v_sd**2*S_wet)
CL = 2*L/(rho*v_sd**2*S_wet)

E = -CL/CD

for i in range(0,len(E)):
    if (math.fabs(E[i]) > 10):
        E[i] = E[i-1]
```

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


fig1 = plt.figure(figsize=(40, 30), dpi=90)
plt.grid()
matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)
sf = 500;
st = 0*sf; fn = 1100*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],D[st:fn],linewidth='1',color = 'r',label
    ='Drag');
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[N]$',fontsize=40)
plt.legend(fontsize=30)
plt.show()

fig2 = plt.figure(figsize=(40, 30), dpi=90)
plt.grid()
matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)
sf = 500;
st = 540*sf; fn = 560*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn], FY_sd[st:fn],linewidth='1',color = 'r',
    label='Fy');
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[N]$',fontsize=40)
plt.legend(fontsize=30)
plt.show()

fig3 = plt.figure(figsize=(40, 30), dpi=90)
plt.grid()
matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)
```

```
sf = 500;
st = 150*sf; fn = 250*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn], -L[st:fn],linewidth='1',color = 'r',
    label='Lift');
st = 600*sf; fn = 700*sf;
plt.plot(t[st:fn], -L[st:fn],linewidth='1',color = 'r');
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[N]$',fontsize=40)
plt.legend(fontsize=30)
plt.show()

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

fig1 = plt.figure(figsize=(40, 30), dpi=90)
plt.grid()
matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)
sf = 1000 # Need to Check this one !!!!!
sf = 500
st = 150*sf; fn = 250*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],v_sd[st:fn],linewidth='1',color = 'b',
    label='V');
st = 600*sf; fn = 700*sf;
plt.plot(t[st:fn],v_sd[st:fn],linewidth='1',color = 'b');
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[m/s]$',fontsize=40)
plt.legend(fontsize=20)
plt.show()


fig1 = plt.figure(figsize=(40, 30), dpi=90)
plt.grid()
matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)
sf = 500;
```

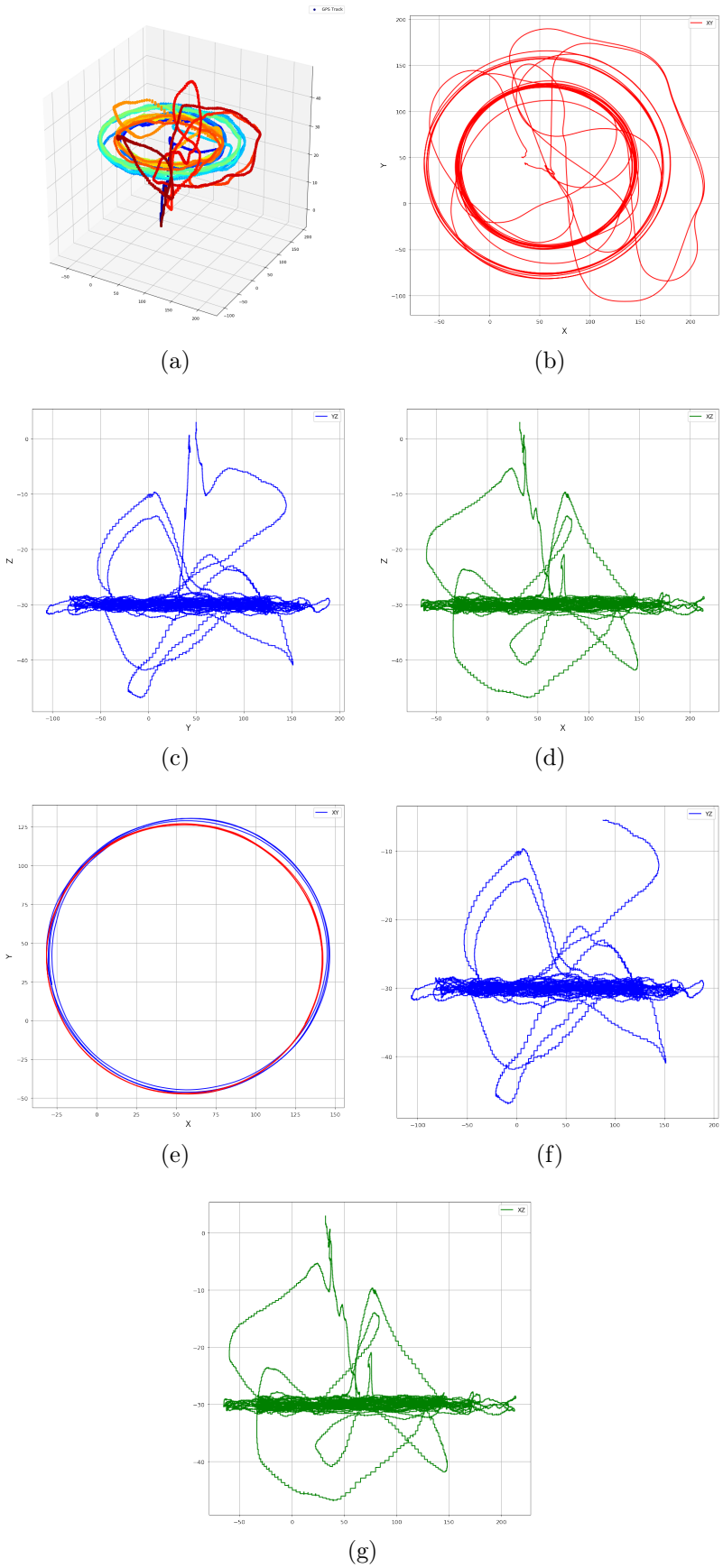
```
st = 105*sf; fn = 1050*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],CD[st:fn],linewidth='1',color = 'g',
    label='Drag coefficient (CD)');
plt.xlabel('$t[s]$',fontsize=40)
plt.legend(fontsize=30)
plt.show()

fig2 = plt.figure(figsize=(40, 30), dpi=90)
plt.grid()
matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)
sf = 500;
st = 105*sf; fn = 1050*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn], CY_sd[st:fn],linewidth='1',color = 'g',
    label='CY');
plt.xlabel('$t[s]$',fontsize=40)
plt.legend(fontsize=30)
plt.show()

fig3 = plt.figure(figsize=(40, 30), dpi=90)
plt.grid()
matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)
sf = 500;
st = 150*sf; fn = 250*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn], -CL[st:fn],linewidth='1',color = 'g',
    label='Lift coefficient (CL)');
st = 600*sf; fn = 700*sf;
plt.plot(t[st:fn], -CL[st:fn],linewidth='1',color = 'g');
plt.xlabel('$t[s]$',fontsize=40)
plt.legend(fontsize=30)
plt.show()
```

```
fig3 = plt.figure(figsize=(40, 30), dpi=90)
plt.grid()
matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)
sf = 500;
st = 105*sf; fn = 1050*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn], E[st:fn],linewidth='1',color = 'b',
    label='Efficiency (CL/CD)');
plt.xlabel('$t[s]$',fontsize=40)
plt.legend(fontsize=30)
plt.show()

#MOMENTS

sf = 500
diff_period = sf
pdot_sd = p_sd.diff(periods=diff_period)
qdot_sd = q_sd.diff(periods=diff_period)
rdot_sd = r_sd.diff(periods=diff_period)

I_xx = 0.00493
I_yy = 0.00532
I_zz = 0.00862
c = 0.135
rho = 1.225
S_wet = 0.0743*2

Mp_sd = pdot_sd*I_xx+q_sd*r_sd*(I_zz-I_yy)
Mq_sd = qdot_sd*I_yy+p_sd*r_sd*(I_xx-I_zz)
Mr_sd = rdot_sd*I_zz+p_sd*q_sd*(I_yy-I_xx)

CMp_sd = 2*Mp_sd[0]/(rho*v_sd**2*S_wet*c)
CMq_sd = 2*Mq_sd[0]/(rho*v_sd**2*S_wet*c)
CMr_sd = 2*Mr_sd[0]/(rho*v_sd**2*S_wet*c)

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

```
fig1 = plt.figure(figsize=(40, 30), dpi=90)
plt.grid()
matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)
sf = 500
st = 0*sf; fn = 1100*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],Mp_sd[st:fn],'r',linewidth='1',label='
    Roll moment');
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[Nm]$',fontsize=40)
plt.legend(fontsize=30)
plt.show()

fig2 = plt.figure(figsize=(40, 30), dpi=90)
plt.grid()
matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)
sf = 500;
st = 0*sf; fn = 1100*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn], Mq_sd[st:fn],'r',linewidth='1',label='
    Pitch moment');
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[Nm]$',fontsize=40)
plt.legend(fontsize=30)
plt.show()

fig3 = plt.figure(figsize=(40, 30), dpi=90)
plt.grid()
matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)
sf = 500;
st = 0*sf; fn = 1100*sf;
N = data.shape[0]
t = np.arange(N)/sf
```

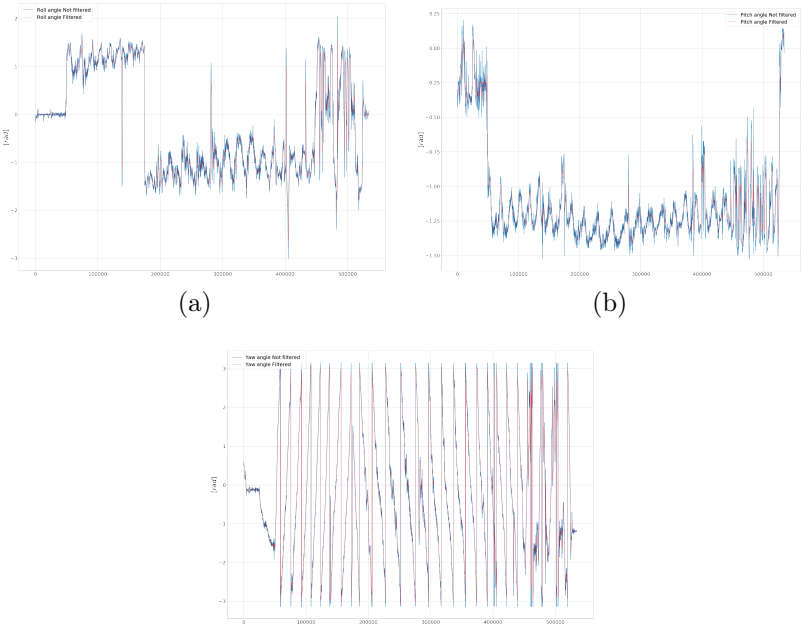
```
plt.plot(t[st:fn], Mr_sd[st:fn],'r',linewidth='1',label='
    Yaw moment');
plt.xlabel('$t[s]$',fontsize=40)
plt.ylabel('$[Nm]$',fontsize=40)
plt.legend(fontsize=30)
plt.show()

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

fig1 = plt.figure(figsize=(40, 30), dpi=90)
plt.grid()
matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)
sf = 500
st = 105*sf; fn = 1050*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],CMp_sd[st:fn],'g',linewidth='1',label='
    Rolling moment coefficient (CMx)');
plt.xlabel('$t[s]$',fontsize=40)
plt.legend(fontsize=20)
plt.show()

fig2 = plt.figure(figsize=(40, 30), dpi=90)
plt.grid()
matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)
sf = 500;
st = 105*sf; fn = 1050*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],CMq_sd[st:fn],'g',linewidth='1',label='
    Pitching moment coefficient (CMy)');
plt.xlabel('$t[s]$',fontsize=40)
plt.legend(fontsize=20)
plt.show()

fig3 = plt.figure(figsize=(40, 30), dpi=90)
plt.grid()
```

```
matplotlib.rc('xtick', labelsize=30)
matplotlib.rc('ytick', labelsize=30)
sf = 500;
st = 105*sf; fn = 1050*sf;
N = data.shape[0]
t = np.arange(N)/sf
plt.plot(t[st:fn],CMr_sd[st:fn],'g',linewidth='1',label='
    Yawing moment coefficient (CMZ)');
plt.xlabel('$t[s]$',fontsize=40)
plt.legend(fontsize=20)
plt.show()
```



(a) (b) (c) (d) (e) (f) (g)

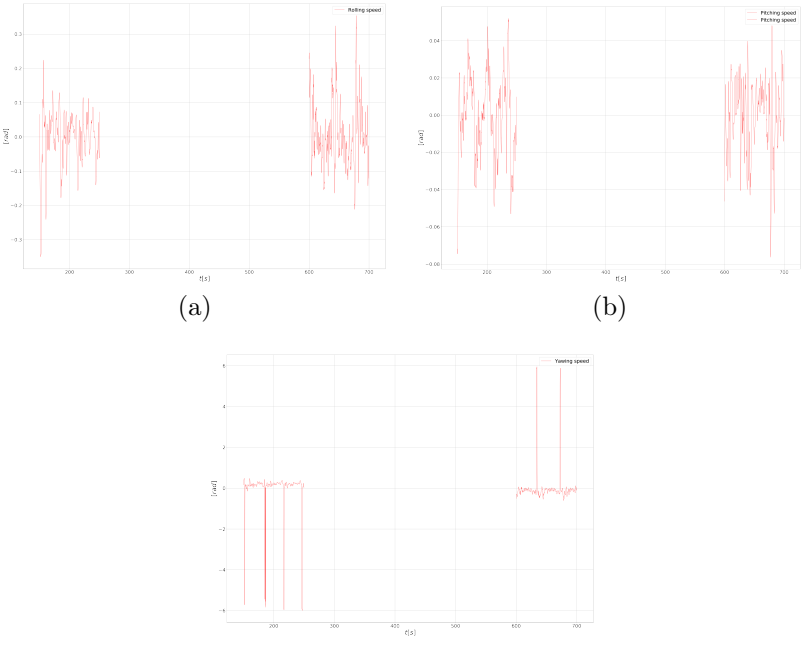
Figure 69: Outdoor flight tracks and details



(a) (b)

(c)

Figure 70: Euler angles filtering comparisons



(a) (b)

(c)

Figure 71: Angular speeds

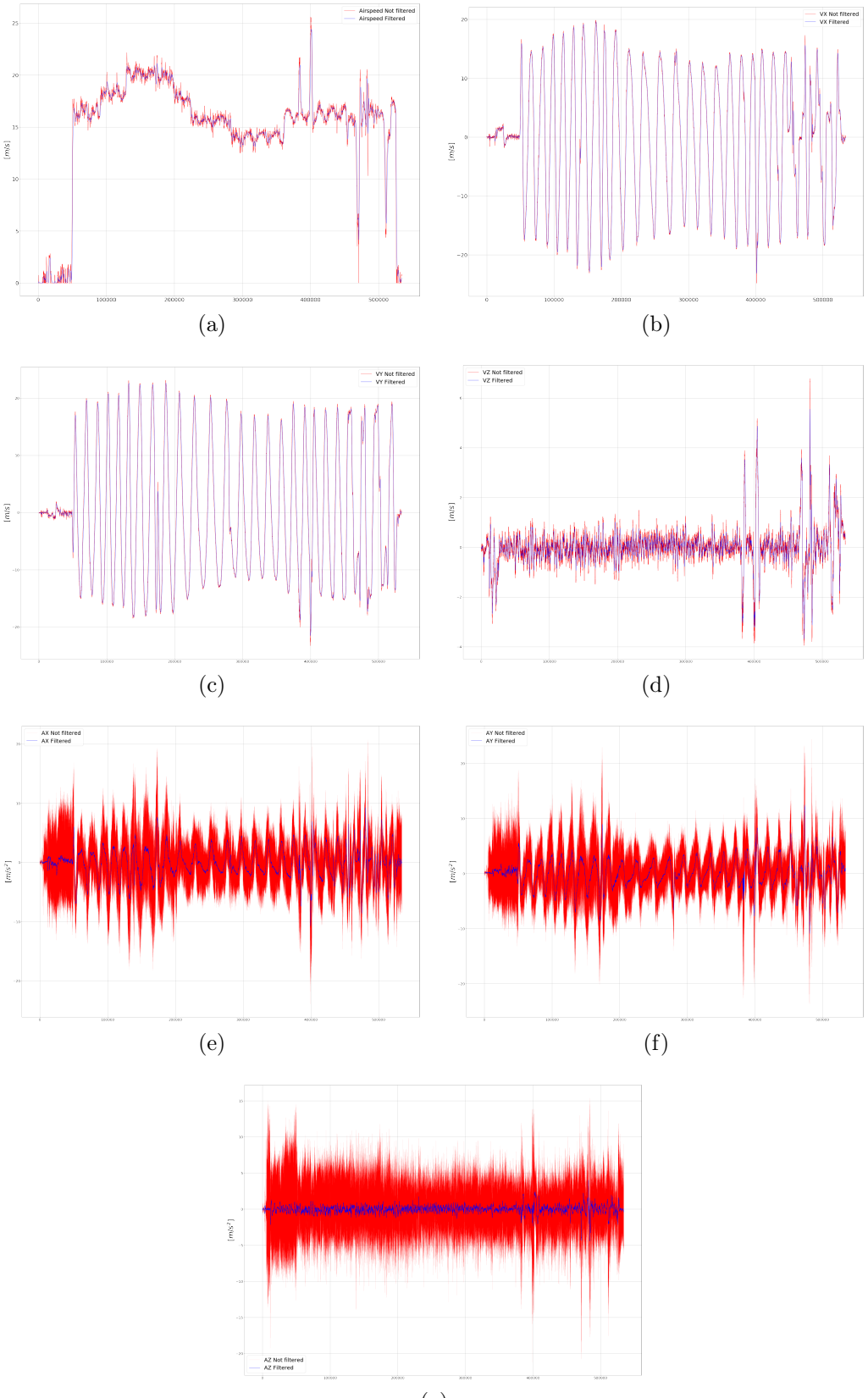


Figure 72: Linear speeds and accelerations filtering comparisons

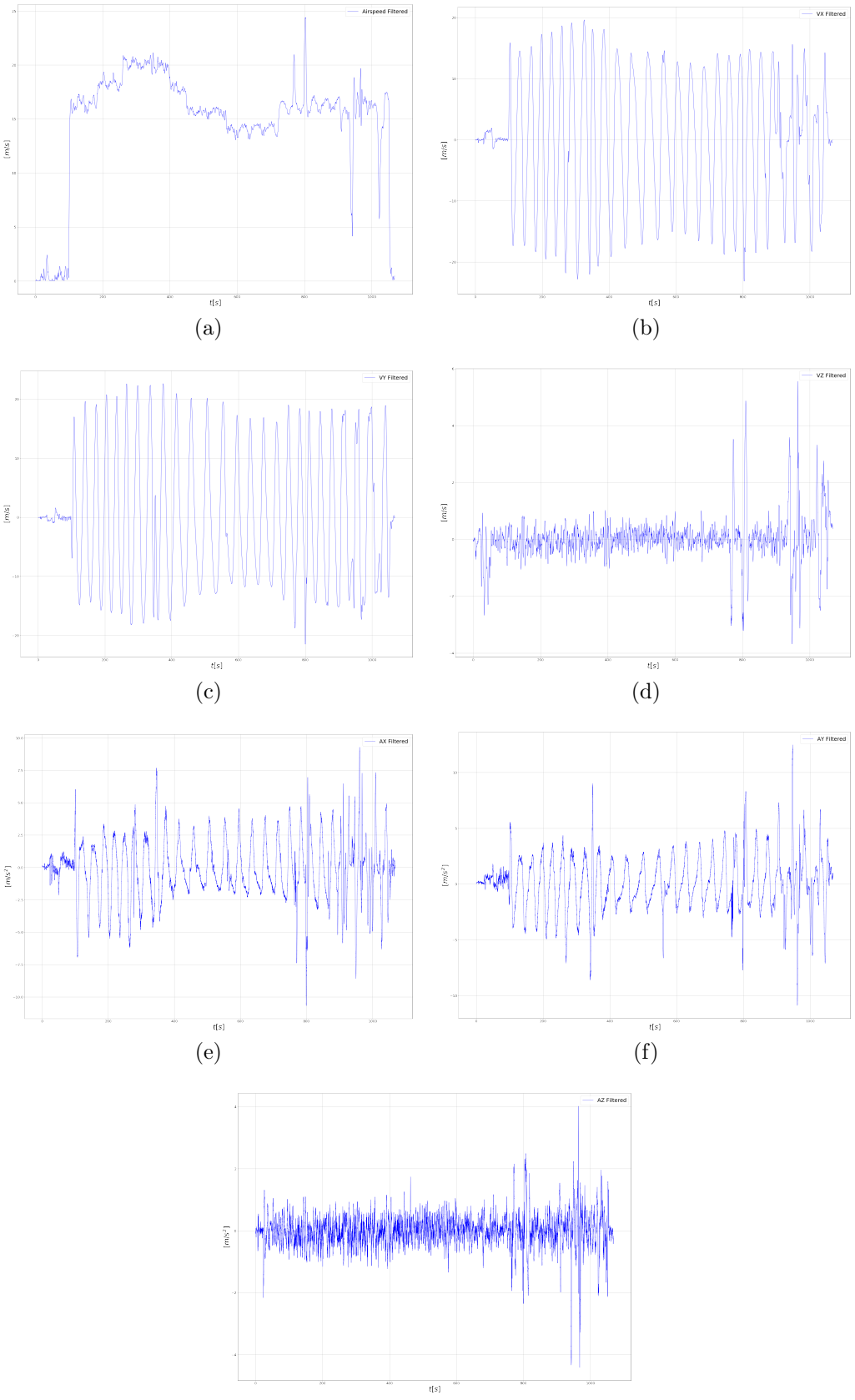


Figure 73: Filtered linear speeds and accelerations

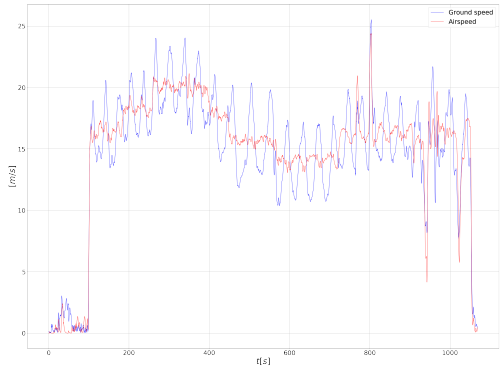
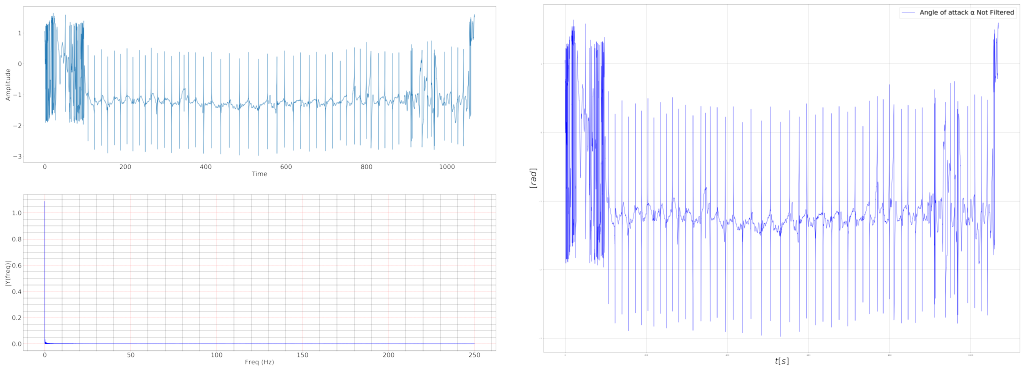
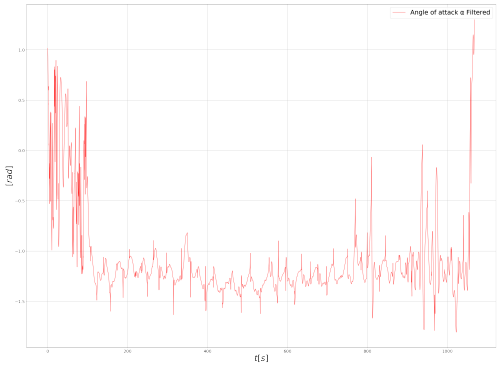


Figure 74: Ground speed vs airspeed



(a) (b)



(c)

Figure 75: Angle of attack filtering process

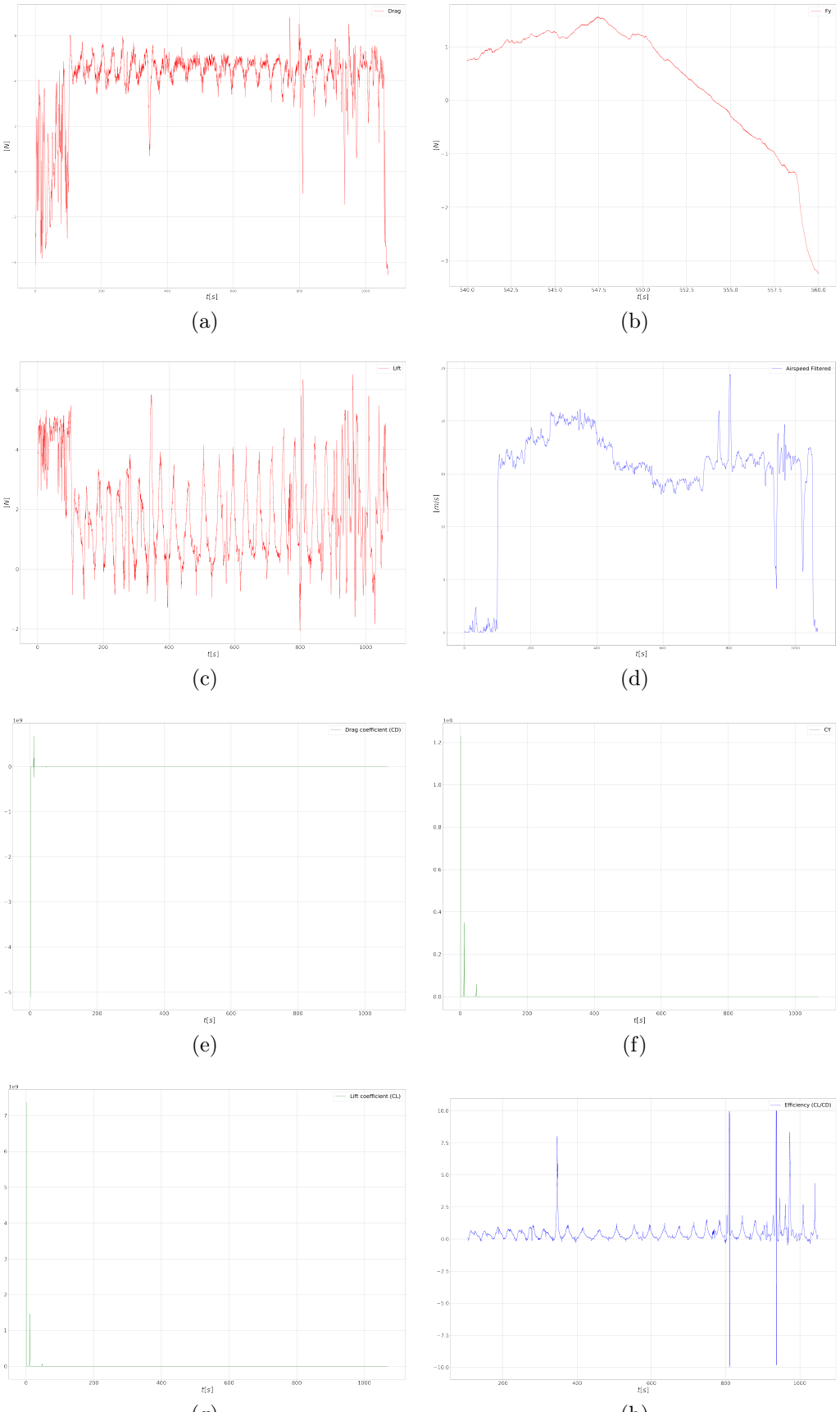
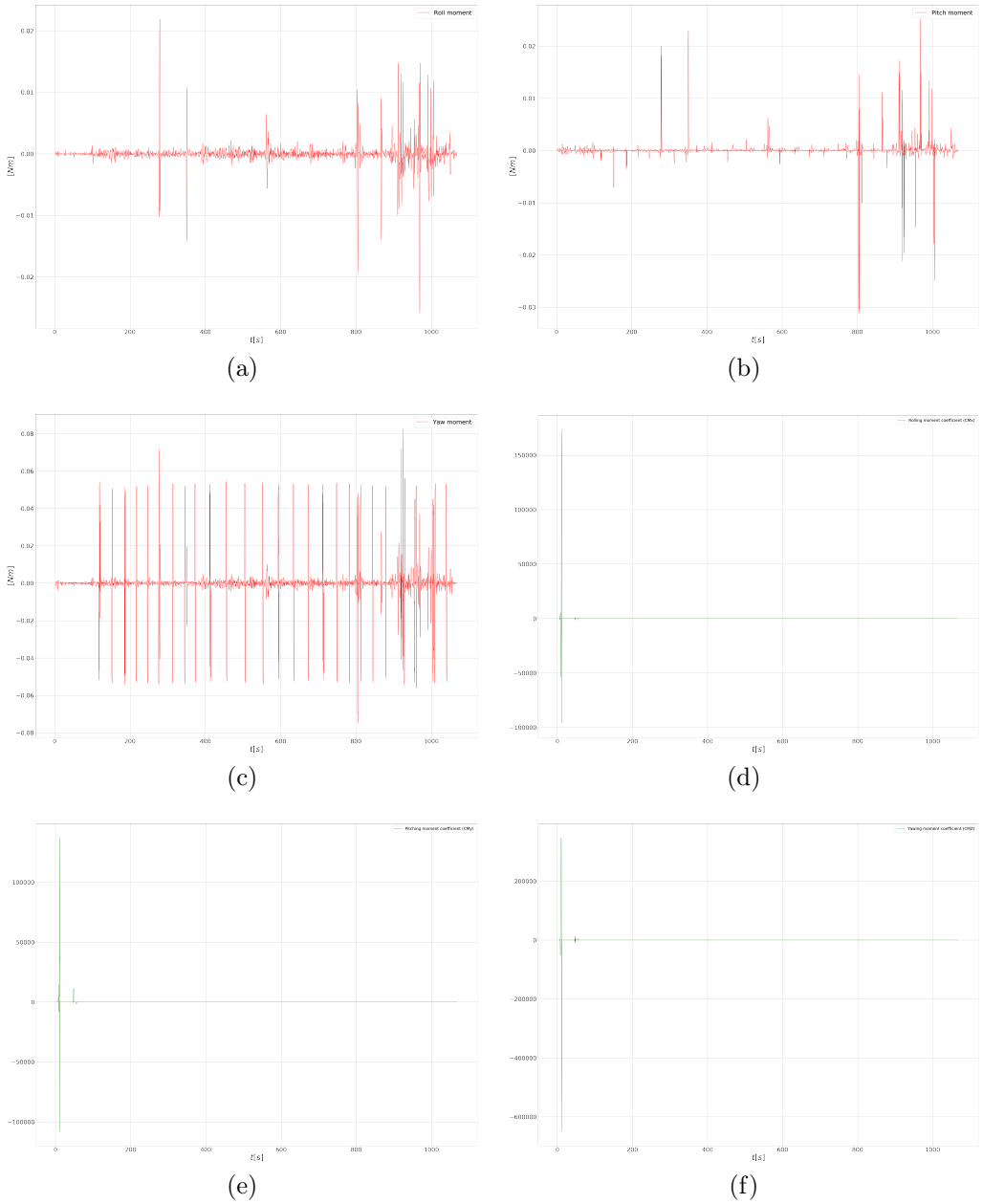


Figure 76: Aerodynamic forces and coefficients



(a) (b) (c) (d) (e) (f)

Figure 77: Aerodynamic moments and coefficients

E Appendix E - Outdoor flight FFT

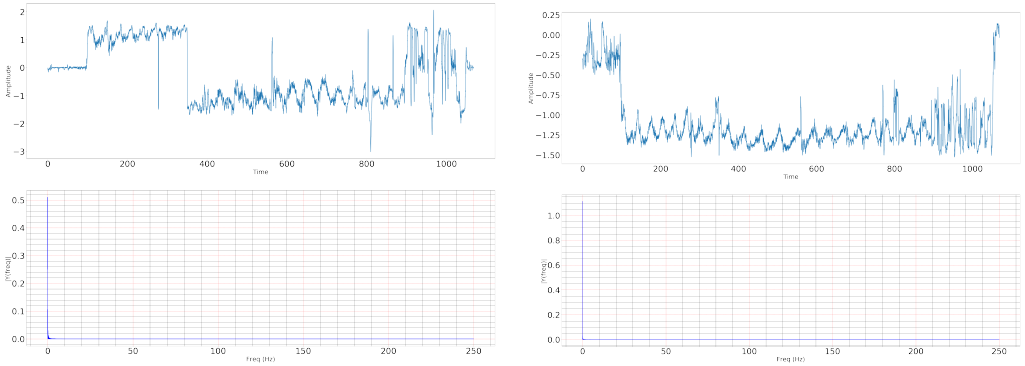
```
#!/usr/bin/env python
from __future__ import print_function, division
import numpy as np
import math
import pandas as pd
import re
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.mlab as mlab
import glob # use this for file O later
from numpy import sin, cos, pi, sqrt, dot
from scipy import stats, optimize
from scipy import linalg as la
from scipy.optimize import leastsq
from scipy.optimize import least_squares
import pdb
from mpl_toolkits.mplot3d import Axes3D
from scipy.interpolate import griddata, interp1d
from scipy import signal
from scipy.optimize import curve_fit
from scipy import interpolate
import scipy as sp

# Read the binary log

file_name = "pprzlog_0099.LOG" #Long flight with different
    airspeeds

dt = np.dtype('i4');
di = np.fromfile(file_name, dtype=dt);

dt = np.dtype('f4');
df = np.fromfile(file_name, dtype=dt);

col = 12+26 ;
exc = np.mod(di.size,col) ;
```

```
row = int((di.size-exc)/col);

di = di[:-exc].reshape(row,col);
df = df[:-exc].reshape(row,col);

d = np.zeros([row,col]);
d[:,0:12]  = di[:,0:12];
d[:,12:]   = df[:,12:];
data = d.copy()

# Sample frequency
sf = 512;
#First order actuator dynamics constant (discrete,
   depending on sf)
fo_c = 0.025

N = data.shape[0]

# Data structure
t = np.arange(N)/sf
counter  = data[:,0]
act      = data[:,1:5] #/9600 # roll-pitch-yaw
quatsp   = data[:,5:9]/2**15 # quat i-x-y-z
airspeed = data[:,11]/2**19
gyro = data[:,12:15] # roll-pitch-yaw
quat = data[:,15:19] # quat i-x-y-z
pos  = data[:,19:22]
speed= data[:,22:25] # NED speed ?
indi_v = data[:,25:29]
accel_sp  = data[:,29:32]
accel_ned = data[:,32:35]
speed_sp  = data[:,35:38]

roll_sd_nf = np.zeros(len(quat))
pitch_sd_nf = np.zeros(len(quat))
yaw_sd_nf = np.zeros(len(quat))

for i in range(0,len(quat)):
    e0 = quat[i,0]
```

```
    e1 = quat[i,1]
    e2 = quat[i,2]
    e3 = quat[i,3]

    #roll (x-axis rotation)
    arg1 = np.arctan2((e2*e3+e0*e1),(+1/2-(e1**2+e2**2)))
    roll_sd_nf[i] = arg1

    #pitch (y-axis rotation)
    arg2 = (+2.0*(-e1*e3+e0*e2))
    if (math.fabs(arg2) >= 1):
        arg2 = copysign(math.pi/2,arg2); # use 90 degrees
   if out of range
        pitch_sd_nf[i] = arg2
    else:
        arg2 = np.arcsin(arg2);
        pitch_sd_nf[i] = arg2

    #yaw (z-axis rotation)
    arg3 = np.arctan2((e1*e2+e0*e3),(+1/2-(e2**2+e3**2)))
    yaw_sd_nf[i] = arg3

def plot_fft(signal,freq):

    Fs = freq;  # sampling rate
    y = signal;
    n = len(y) # length of the signal
    duration = n/Fs; # signal duration
    Ts = 1.0/Fs; # sampling interval
    t = np.arange(0,duration,Ts) # time vector

    # ff = 10;   # frequency of the signal
    # y = np.sin(2*np.pi*ff*t) + np.sin(2*np.pi*ff*t*10)

    k = np.arange(n)
    T = n/Fs
    frq = k/T # two sides frequency range
    frq = frq[range(int(n/2))] # one side frequency range

```

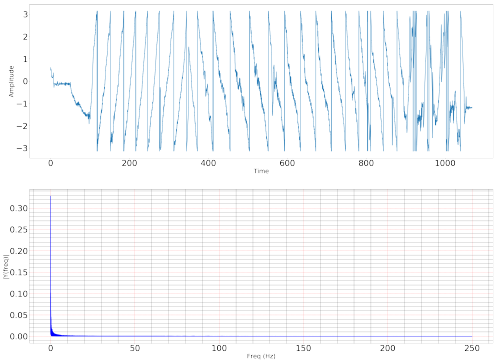
```
    Y = np.fft.fft(y)/n # fft computing and normalization
    Y = Y[range(int(n/2))]

    fig, ax = plt.subplots(2, 1,figsize=(40,30),dpi=90)
    matplotlib.rc('xtick', labelsize=40)
    matplotlib.rc('ytick', labelsize=40)
    ax[0].plot(t,y)
    ax[0].set_xlabel('Time',fontsize=30)
    ax[0].set_ylabel('Amplitude',fontsize=30)
    ax[1].plot(frq,abs(Y),'blue') # plotting the spectrum
    ax[1].set_xlabel('Freq (Hz)',fontsize=30)
    ax[1].set_ylabel('|Y(freq)|',fontsize=30)

    # Turn on the minor TICKS, which are required for the
   minor GRID
    ax[1].minorticks_on()
    # Customize the major grid
    ax[1].grid(which='major', linestyle='--', linewidth='
   0.5', color='red')
    # Customize the minor grid
    ax[1].grid(which='minor', linestyle='-', linewidth='
   0.5', color='black')
    plt.grid(True, lw = 2, ls = '--', c = '.5');plt.show()

print('roll_sd_nf')
plot_fft(roll_sd_nf,500)
print('pitch_sd_nf')
plot_fft(pitch_sd_nf,500)
print('yaw_sd_nf')
plot_fft(yaw_sd_nf,500)

print('speed[:,0]')
plot_fft(speed[:,0],500)
print('speed[:,1]')
plot_fft(speed[:,1],500)
print('speed[:,2]')
plot_fft(speed[:,2],500)

print('airspeed')
```

```
plot_fft(airspeed,500)

print('accel_ned[:,0]')
plot_fft(accel_ned[:,0],500)
print('accel_ned[:,1]')
plot_fft(accel_ned[:,1],500)
print('accel_ned[:,2]')
plot_fft(accel_ned[:,2],500)
```



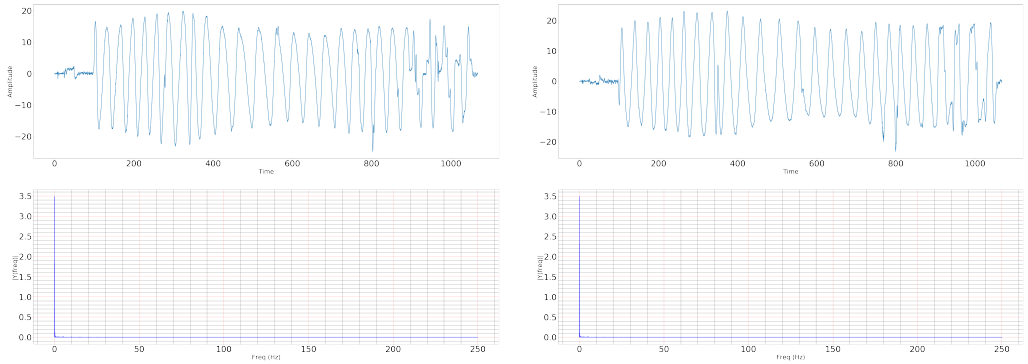
(a)

(b)



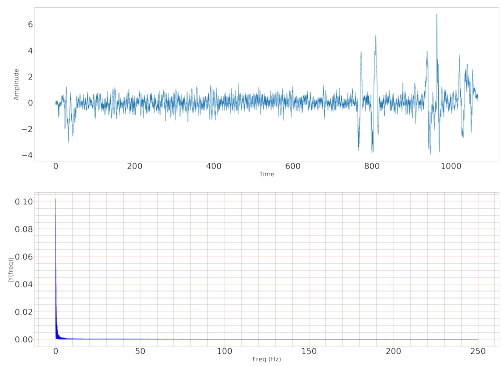
(c)

Figure 78: Euler angles FFT output

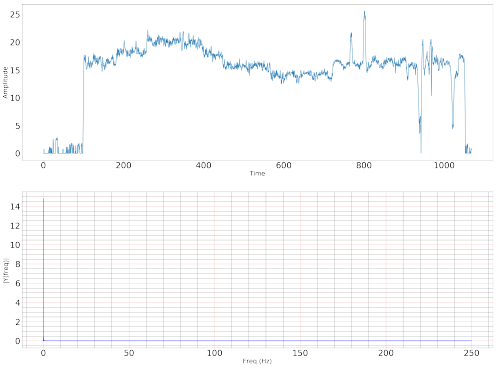


(a)

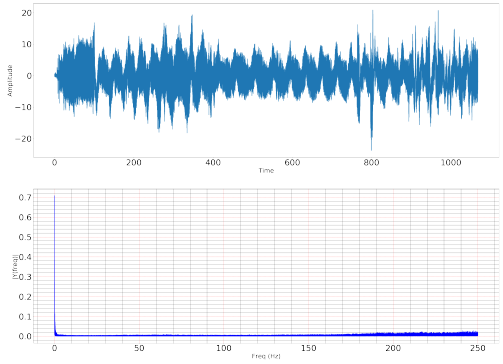
(b)



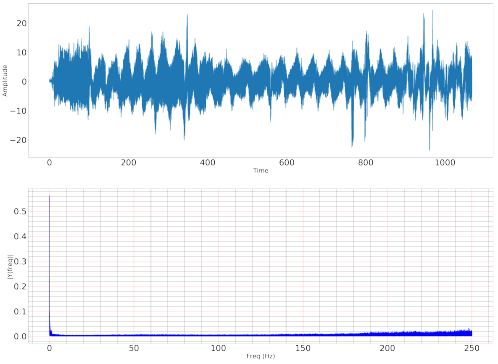
(c)



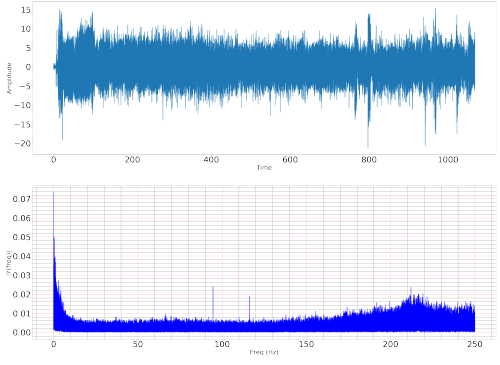
(d)



(e)



(f)



(g)

Figure 79: Linear speeds and accelerations FFT output