Politecnico di Torino

Master's Degree Thesis
In
Aerospace Engineering

Device for non-invasive Flight Test Instrumentation (FTI): hardware & software

Supervisors:
Prof. Manuela Battipede
Politecnico di Torino

Dr. Francesco De Vivo
HighTek S.r.l.

Candidate:
Samuel Lazzaro

Academic Year 2019/2020

Summary

List of Figures

1 Flight Test

1.1 Introduction

The products of the aerospace world are continuously subjected to checks and tests necessary to determine their correct functioning. The first tests are carried out on the prototypes of the entire aircraft or even just of one of its systems (e.g. propulsion system) and are performed at the end of product development to validate and certify its correct functioning.
During the operational life of the aircraft, further flight tests may be carried out, for example to verify that no functional problems have arisen. When changes are made in certain parts, it is necessary to recertify the aircraft by carrying out a series of flight tests that allow to obtain the necessary data to be analyzed and compared with those prior to the changes or with the Regulations, in order to verify the correct functioning of the aircraft and thus to obtain the new certification.

Carrying out a flight test campaign is not easy; in fact, in addition to requesting a Permit to Fly issued by the competent national authority or by an Approved Organization (DOA/POA), it is necessary to have the adequate FTI (Flight Test Instrumentation) for the flight tests to be performed. The crew that has to carry out the flight tests must also be made up of members with specific skills.
There is a document called FTOM (Flight Test Operations Manual) that describes the flight test organization's involvement in the process to issue a Permit to Fly and describes the organization's policies and procedures in relation to flight test. In this manual, in fact, there must be a chart that represent the organizational structure and the links between the different professional roles whom participate in the flight test activities.
There must be also a list of the essential qualifications and a description of each team member's roles and responsibilities for that flight test category, in order to help the company to ensure that it is composed of qualified personnel to perform those particular flight test operations.

An indispensable professional figure is that of the Safety Manager, who must collect and analyze hazards and maintain a register of risks, hazards and risk controls mitigations.
Each flight test organization must develop a Safety Management System (SMS) in order to consider traditional safety risks and to manage them in a systematic way.
A flight test Risk Management is also included in the FTOM, which is complementary to the Safety Management, but they are not the same thing. In fact, the SMS manages risks common to all flight activities (e.g. bird strike, mid-air collision), while the Risk Management manages all the risks associated with a particular flight test [1].

The duration of a flight test campaign is highly variable because these tests can relate to a single new system of an existing aircraft or also to a complete development and

certification of a new aircraft. Therefore, it can vary from a few weeks to many years [2].

Flight tests can be performed on both civil and military aircraft.
First of all, it is necessary to define which system or part of the aircraft must be tested and which tests must be performed. The content of the flight test determines the flight test category, and the latter determines the required competence of the flight test crew.

1.2 Flight Test Categories

The Flight Test categories are defined in Appendix XII to Part-21 and they are:

- Category 1 flight test
  - "*Fixed-wing aircraft: VMCG, VMU, spinning, initial stalling, or for rotary-wing aircraft: H/V diagrams and Category A engine failures.*
  - *Where encounter of surprising or even hazardous flight characteristics can be expected.*
  - *Upon determination, aircraft handling and performance in conditions where at least one of the following parameters is approaching the actual limits of the aircraft envelope: altitude, attitudes, weights, CG, speed/Mach, stalls, temperature, engine and airfoil performance.*
  - *Where the embodiment of new systems is anticipated to significantly affect the aircraft's handling or performance characteristics.*
  - *When the crew of the chase aircraft has the duty to assist the test aircraft crew in recovering from a critical flight situation (i.e. assist the spinning aircraft crew in assessing the spin or triggering recovery actions)."* [3]

- Category 2 flight test
  - "*The flight test envelope has already been opened and it has been demonstrated that the general behavior of the aircraft is adequately safe and there are no unsafe flight characteristics.*
  - *All-engines-operating climb performance.*
  - *Cruise performance.*
  - *Static stability demonstration.*
  - *Function and reliability flights.*
  - *Systems tests of autopilot or guidance/warning systems such as Terrain Awareness and Warning System (TAWS) or Airborne Collision Avoidance System (ACAS), when the modes themselves are tested, requiring operating the aircraft by deviating from the standard operational procedures. Additionally, in the case of embodiment of such systems on an already certified aircraft, when the system integration in an existing*

cockpit requires a more global crew procedure assessment - for example, when the system has been integrated in cockpit screens and a centralized warning system which requires a new cockpit procedure assessment (note that some system tests may fall under Category 4; see below).” [3]

- Category 3 flight test
  This category concerns all the flight tests performed on a new aircraft that already has a Type Certificate (TC) or a Supplemental Type Certificate (STC) but which behavior is not yet known, so unexpected failure can occur which could not be described in the Aircraft Flight Manual (AFM). Therefore, it is necessary to perform these flights that are commonly referred to as production flight tests.
  If an aircraft does not have a TC or STC, any flight will be Category 1, 2 or 4 according to classification criteria.
  If the flight test of an aircraft with a TC or STC requires flying outside the AFM limitations, this flight should be considered as Category 1 or Category 2 flight test.

- Category 4 flight test
  These flights are those required by a DOA (Design Organization Approval) to demonstrate compliance with the airworthiness requirements of “not yet approved data”:
  - *cabin conversion;*
  - *zonal drying system installation;*
  - *Emergency Locator Transmission (ELT) installation;*
  - *new cabin installation;*
  - *cabin aircraft location pictorial system installation;*
  - *new entertainment system installation;*
  - *SATCOM and telephone installation;*
  - *new radio equipment installation.* [3]

Once the flight test category has been established, crew members are chosen according to it: a Flight Test Organization required team members with different skills, as test pilots, flight test engineers, designers, mechanics, certifying staff and safety officer [4].

Subsequently, the Flight Test Engineer prepares a test plan containing the various maneuvers to be carried out during the flight.

1.3 Flight Test Data

Once the maneuvers to be carried out have been established, the aircraft is appropriately instrumented. The FTI (Flight Test Instrumentation) used must be suitable for the type of data to be collected, which can be [2]:

- accelerations in all six degrees of freedom;
- aircraft attitude, angle of attack and sideslip angle;
- aircraft controls deflection (stick/yoke, rudder pedals, throttle position);
- engine performance parameters;
- noise levels;
- internal temperatures;
- structural loads.

Once the aircraft is instrumented, it is possible to begin ground and flight tests.

2 Requirements

In this thesis project two different applications have been dealt with, each of which having its own requirements.

1st Application Requirements:

- FTI easy to transport and to mount on the aircraft;
- minimal use of external power sockets;
- use of 1 to 4 sensors;
- possibility to view in real time the images taken by a camera and the accelerations trends and values in the time domain;
- possibility to perform a Fast Fourier Transform (FFT) in real-time during the flight without having to use MATLAB;
- possibility to start and stop saving data in certain time windows using appropriate buttons.

2nd Application Requirements:

- use a sensor capable of detecting the main dynamic parameters;
- possibility to view in real time the time histories of the main dynamic parameters;
- possibility to start and stop saving data in certain time windows using appropriate buttons;
- identify or implement the algorithms necessary to obtain information about the natural frequency f_n and damping ratio ζ of each parameter and / or dynamic mode.

3 Flight Test Instrumentation (FTI)

3.1 Data and Sensors

In the aeronautical field, flight test campaigns are often carried out with the aim of obtaining data that can be used for certification, for any modifications or for the development of flight simulators of that type of aircraft.
In order to carry out a flight test campaign, it is necessary to have the appropriate Flight Test Instrumentation (FTI) based on the data type to be measured and on the flight tests that must be carried out.

In order to certify an aircraft modification, some of the data to be collected are:

- accelerations along X, Y, Z axes: to obtain the vibration levels to which a certain point of the structure is subject.
- Euler's angles, angular velocities and angular accelerations: to perform a dynamic analysis.

Specific sensors are required to detect these types of data: accelerometers are usually used to detect accelerations, but it is necessary to distinguish between the different types available on the market and choose which of them is the most appropriate for the case study.
Accelerometers are mainly divided into:

- Monoaxial accelerometers: allow to detect accelerations along a given axis (X, Y or Z) based on their positioning. Each one required only one readout instrument input channel.

Figure 1: PCB monoaxial accelerometer

- Triaxial accelerometers: allow to detect accelerations along all three axes (X, Y and Z). Each one required three readout instrument input channels, one for each axis. Moreover, their cost is more than twice than that of a monoaxial.

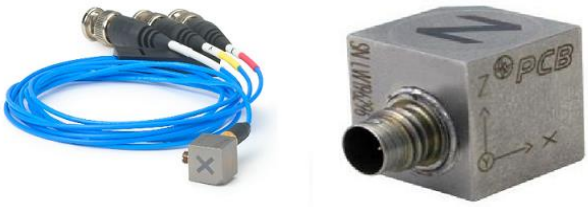


Figure 2: PCB triaxial accelerometer

Considering the probability of an accelerometer breaking due to external causes and the significant cost difference between these two types, it is preferable to have three monoaxial rather than a triaxial one, to still have 2 other accelerometers available.

There are different types of accelerometers based on their operating principle, for example there are strain gauge, LVDT type accelerometer, laser accelerometer, MEMS (Micro-Electro-Mechanical Systems) accelerometer, but a right compromise between required quality and price is represented by piezoelectric accelerometers.

A piezoelectric accelerometer generates an electrical signal via a piezoelectric crystal based on the compression it undergoes due to the inertia force generated by a mass located on that crystal.

There are two different types of piezoelectric sensors [5]:

- **Charge Output Sensors**: they are piezoelectric sensors without built-in electronics, with a high impedance output signal and they usually require external charge or voltage amplifiers for signal conditioning.
- **Internally Amplified Sensors**: they are piezoelectric sensors with built-in electronics, integrated circuits, and with a low impedance output signal. These types of sensors are called ICP® (Integrated Circuit Piezoelectric, is a registered trademark of PCB Group, Inc.) or IEPE (Integrated Electronics Piezo-Electric).

3.2 Charge Output Sensors

"Charge output sensors have the advantage of being able to operate under high temperature environments and withstand up to +281°C. The output signal generated by the piezoelectric signal is extremely sensitive to corruption from various environmental factors, so low-noise cabling must be used to reduce radio frequency interface (RFI) and electromagnetic interference (EMI)." Moreover, it is necessary to use tie wraps or tape in order to reduce the noise due to the triboelectric effect, that is the noise generated by cables motion [1].

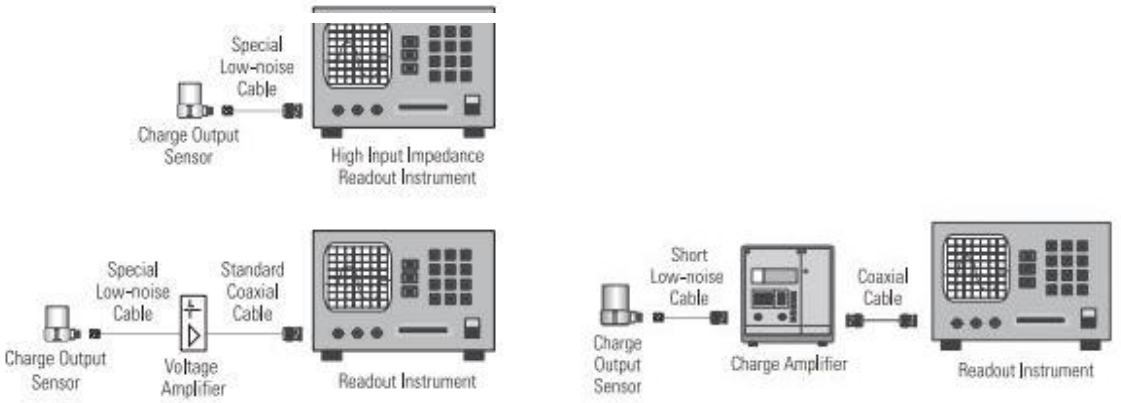


Figure 3: Typical charge output sensor system

The output signal of a charge output sensor is characterized by a high impedance, therefore, in order to correctly analyze the signal, it is necessary either to use a high input impedance readout instrument or an in-line voltage and charge amplifier to convert the output signal in a low impedance one.
Another disadvantage is that the high impedance of the output signal causes a loss of the signal quality directly proportional to the cable length, therefore short low-noise cables must be used.
This kind of cable has a graphite lubricant embedded in the dielectric layer in order to minimize friction and generation of electrostatic charge (triboelectric effect) generated by cable motion.

PE (PiezoElectric) accelerometer resolution is not generally specified on a datasheet because it depends on the noise generated along the cables and on the amplifier gain.

Moreover, high impedance circuits required training and expertise to understand, operate and maintain them. In fact, all high-impedance components must be kept clean and dry, because their contamination due to adverse environment causes noise, loss of signal quality and loss of low frequency response.

3.3 ICP® Sensors

ICP® is a term that uniquely identifies PCB's piezoelectric sensors with built-in electronics. ICP or IEPE sensors have many advantages over charge output sensors:

- Low impedance output signal (<100 ohms): the signal quality does not depend on the cable length, therefore long cables can also be used without increase in noise, loss of resolution, or signal attenuation. Signal quality it is not sensitive to adverse environment because IEPE sensors are resistant to contamination.

- They are less sensitive to electrical interference (RFI and EMI) thanks to their low impedance signal output, so it is not necessary to use low-noise cable, but standard coaxial cables are enough. This implies a significant cost reduction.
- *"Low per-channel cost because sensors require only low-cost, constant current signal conditioners and ordinary cables."* [5] PE (PiezoElectric) and ICP sensors have essentially the same cost, but the per-channel cost of the ICP system is significantly lower because low-noise cables and charge amplifiers are not required.
- They require less electrical power consumption.
- ICP sensor resolution is specified on the datasheet.
- They are easier to use, so less operator expertise, training and attention is required compared to charge output sensors and high impedance circuits.

The only limit of IEPE sensors is that they cannot be used for operation under temperatures environments outside the range -320°F < T < 325°F (-195.6°C < T < 162.8°C).

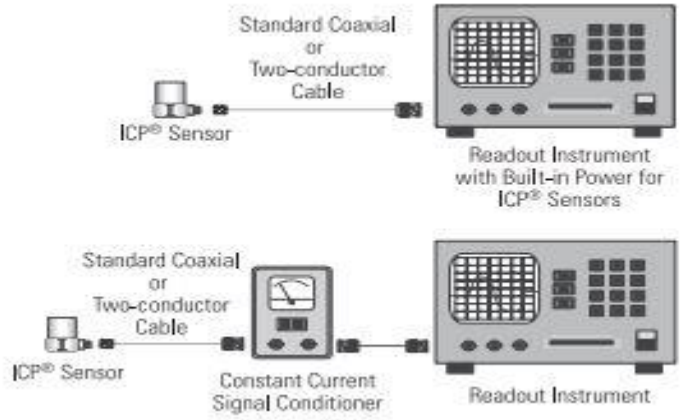


Figure 4: Typical ICP sensor system

Therefore, for these reasons, it is preferred to use IEPE sensors.
All IEPE sensors require a constant current power source for proper operation: they usually require a power supply current between 2mA - 4mA, but sometimes this range could be extended to 0.5mA - 20mA depending on the sensor application.
It is therefore necessary to check the compatibility between the current required by the IEPE sensor and that supplied by the signal conditioner or by the readout instrument with an IEPE interface.

3.4 What is an oscilloscope

An oscilloscope is an electronic test instrument that allows the user to view on a 2D graph the noise and vibration signals obtained from one or more appropriate sensors. The graph is displayed on the oscilloscope monitor or on a PC and the various trends can be viewed both in the time domain and/or in the frequency domain according to the user's needs.

There are different types of oscilloscopes based on functionality and workplace.
There are bench oscilloscopes used very often in laboratories as they are very bulky, or portable oscilloscopes that have fewer functions but that can be easily transported and used in different workplaces.

The sensors are connected to the oscilloscope via BNC coaxial cables.

The oscilloscopes can be powered via a power outlet, via batteries or via the USB port of a PC.
If the oscilloscope can be connected to a PC via a USB port, there are usually software that allow real-time visualization of the various graphs.

Figure 5: Bench Oscilloscope and Portable Oscilloscope

Figure 6: USB Oscilloscope

In this application, having to carry the oscilloscope on board an aircraft, it is preferable to use an USB oscilloscope because it is easily transportable, not bulky and that does not require a power supply from a power outlet.

3.5 Signal Conditioner or Oscilloscope IEPE

Two different devices can be used to feed an ICP sensor: a signal conditioner or an oscilloscope with an IEPE interface.
A signal conditioner is a necessary device to provide the correct current intensity to the IEPE sensors. Instead, an oscilloscope with an IEPE interface allows a direct connection to the IEPE sensors, without having to use a signal conditioner.

There are different types of signal conditioners: they can be powered by battery or by an external 32-38V DC power supply. In the latter case it is necessary to use a DC/DC converter since helicopters usually have a 28V DC power socket. Then, a readout instrument as an oscilloscope or a spectrum analyzer is necessary to save the data and to display them, via software, on a PC in time or frequency domain. Normally the readout instrument can be connected to the PC via USB or ethernet cable.
Currently, many instruments available on the market can be used in both oscilloscope or spectrum analyzer modes, thus allowing the user to view the data in the domain he prefers or sometimes even simultaneously in both time and frequency domains.

For each channel of the oscilloscope, the signal conditioner needs two channels, one for the sensor input and the other for the oscilloscope. For this reason, battery-powered signal conditioners, having smaller dimensions as they are usually portable, have a maximum of three channels for the oscilloscope. However, one of the requirements is to have four channels so that four monoaxial sensors can be used simultaneously.

Figure 7: PCB Signal Conditioner 4-channels line-powered

Figure 8: PCB Signal Conditioner 3-channels battery-powered

In this case it is also preferable not to use a signal conditioner powered by an external DC power supply in order to not use a DC/DC converter.

To avoid having to use a signal conditioner, it is possible to use an oscilloscope with an IEPE interface, even if there are not many available on the market.
One of the available models is the PicoScope 4224 IEPE, a 2-channels oscilloscope developed by Pico Technology: it is directly connected to the PC and powered by a USB cable and it can be used in both IEPE and normal mode.
Its only disadvantage is that it has only two channels, therefore two PicoScope 4224 IEPE are necessary for the flight test activities with four IEPE sensors. The two oscilloscopes must be synchronized to see the various graphs in real-time. To do this synchronization is necessary to use a shared trigger signal, otherwise the scopes will be running to separate internal clocks started independently.

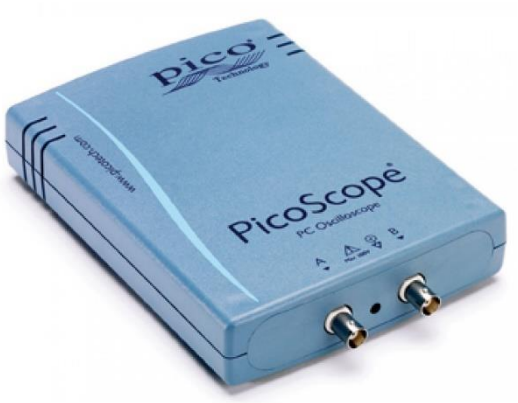


Figure 9: PicoScope 4224 IEPE - USB Oscilloscope

Nevertheless, this solution is preferable as it is not necessary to use a signal conditioner, half the number of cables are necessary because the sensors are directly connected to the oscilloscope, consequently also the reliability of the instrumental equipment is greater and maintenance costs are lower and, moreover, no battery (which could suffer from excessive temperatures) and no external power supply are needed.

In terms of acquisition costs, the two solutions are not very different, consequently the choice falls on the reliability and compactness of the instrumental equipment.
The PicoScope supplied software is compatible with Windows, Linux and macOS operating systems, and it has also the Software Development Kit (SDK).

3.6 Software Development Kit (SDK)

A **Software Development Kit** (**SDK**) is a collection of software development tools in one installable packet [6].
An SDK is made up of libraries and codes that can be written in different programming languages (e.g. Java, C, C++, etc.). These codes allow the programmer to create his own software.
In this application the SDK allows the programmer to directly access the data measured by the sensor and to use and save them as he prefers.

3.7 Basic Instrumentation used

The basic instrumentation used during flight tests is composed by:

- Oscilloscope;
- Signal Conditioner;
- ICP accelerometers.

The main problems that a Flight Test Engineer encounters in using this equipment are:

- both oscilloscope and signal conditioner need to be powered by a power outlet;
- this FTI is too bulky;
- too many cables are needed to connect the various instruments, so there is a higher probability of failure of one of them;
- connecting and disconnecting the various devices requires a certain amount of time;
- data analysis (e.g. Fast Fourier Transform) cannot be performed during the flight.

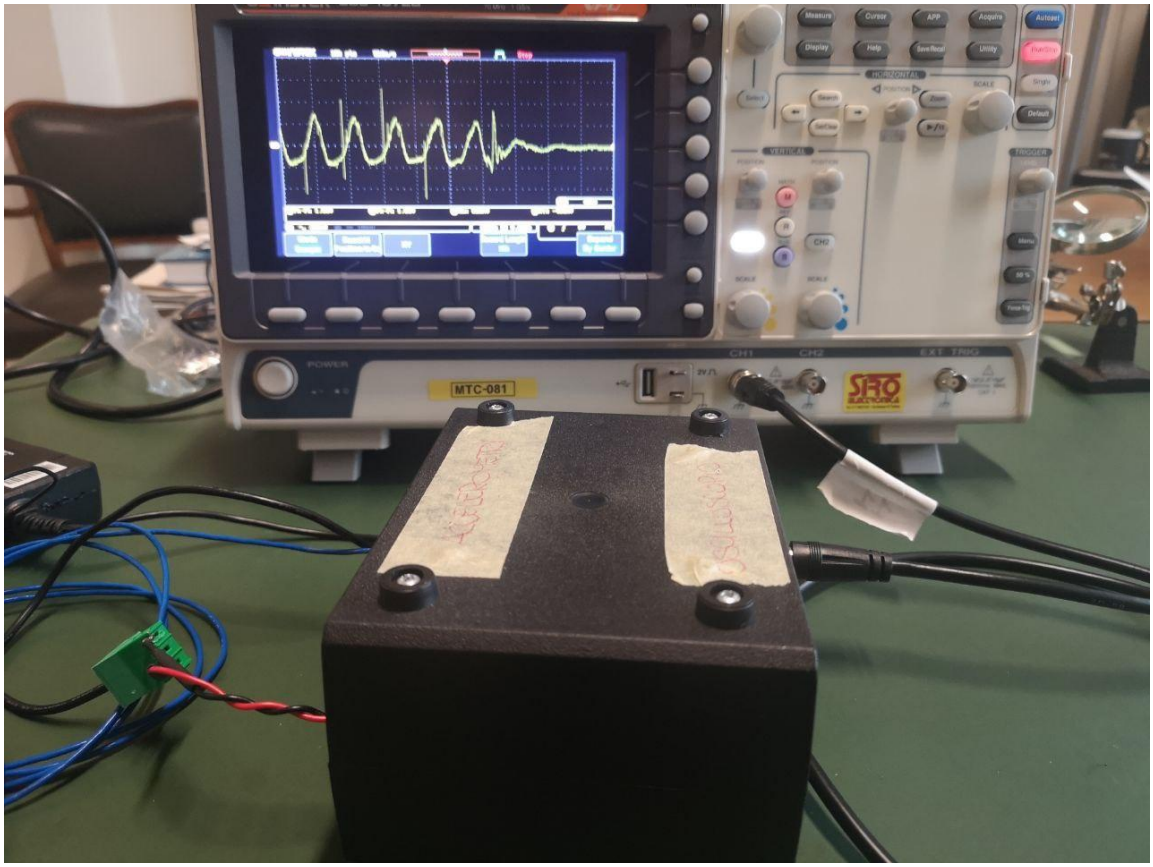


Figure 10: Bench Oscilloscope, Signal Conditioner and IEPE sensors

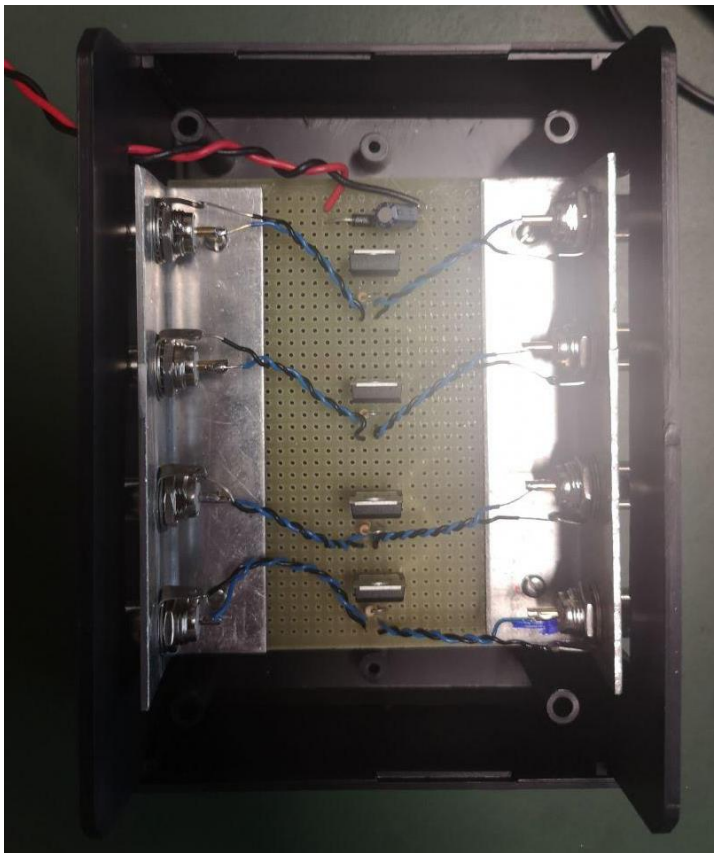


Figure 11: Signal Conditioner internal part

In the figure above there is the internal part of the 4-channel signal conditioner used: care must be taken to use it correctly, i.e. the sensors channels must not be confused with the oscilloscope channels. So, this signal conditioner has 4 input-channel (sensors) and 4 output-channel (oscilloscope).

Figure 12: Miniature DeltaTron monoaxial accelerometer type 4518-003 - Brüel & Kjær

The sensor used for the 1st application is a Miniature DeltaTron monoaxial accelerometer type 4518-003 made by Brüel & Kjær. DeltaTron is the Brüel & Kjær's proprietary name for IEPE accelerometers. Another name used by Brüel & Kjær is CCLD

(Constant Current Line Drive), while other proprietary names for this type of accelerometers are ISOTRON®, PIEZOTRON® and ICP® [7].

This sensor detects a certain voltage value and transmits it to the oscilloscope. Each oscilloscope has a certain resolution: if it has an 8-bits resolution, there are 256 (2^8) possible values that must then be divided between positive and negative values. These values are transcribed in a *.csv* file that can be saved on a suitably formatted USB key. To obtain the acceleration value in *g*, a conversion must be performed: first of all, the values present in the *.csv* file must be converted into volts [*V*], then a conversion factor *V*/*g* is used which allows to obtain the final acceleration value in *g*.

The formula to perform the conversion is the following:

$$acceleration\ [g] = csv\ value \cdot \frac{vertical\ scale \cdot 8}{2^{bit\ acq} \cdot sensor\ sensitivity}$$

where:

- $csv\ value$ is the acceleration value saved on the *.csv* file;
- $vertical\ scale$: is equal to 0.5*V*;
- $bit\ acq$: is the oscilloscope resolution;
- $sensor\ sensitivity$: is the *V*/*g* conversion factor. For the 4518-003 it is equal to 0.1 *V*/*g*.

4 1st Application

The aim of the 1st application is to use a new and adequate FTI in order to verify if a certain part/component of the helicopter is subjected to acceptable accelerations and that these are compliant with the Regulations.
In fact, when a modification is made on board a helicopter, it could be necessary to recertify the helicopter itself as it is necessary to verify that this modification does not cause unacceptable vibrations.

In order to perform this type of analysis, first of all, it is necessary to identify the new hardware components to be used and to write the software code that will be implemented on them.

4.1 First test: PCB USB Signal Conditioner

Initially, a first test was carried out using a PCB 2-channel USB signal conditioner which does not require an oscilloscope to detect and output the accelerations detected by the sensors.

Figure 13: PCB model 485B39 (USB signal conditioner)

The advantage of this product is that it is very small, therefore easily transportable, but it has the disadvantage of not having an SDK. In order to display the accelerations on the screen, i.e. the curves in the time domain or in the frequency domain, in fact it is necessary to use third-party software, which sometimes can also be paid.

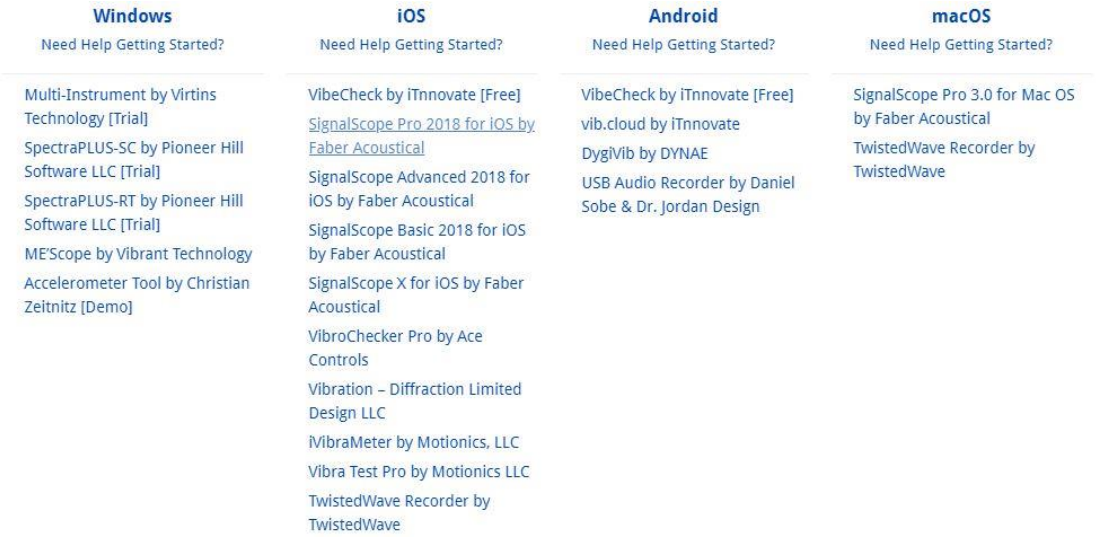


Figure 14: Software List

Using these software, it is also possible to create a text output file in which all the values of the detected accelerations are written, but in this way the accelerations could be used only in a post-processing phase and not during the processing phase. Moreover, there are no software compatible with Linux.
Therefore, not being able to access the detected data at any time, the user is forced to use other software to view the data.
A goal of this application, however, is precisely to be able to create an own software through which the user can view the various curves in real time.
In order to create this software, it is essential that the detection instrument used has an SDK. In this way, the software programmer work is simplified.

4.2 Second test: Odroid, blueFOX3 camera & Spatial

A second test was carried out using the following instrumentation:

- **Odroid-N2**: is a new generation single board computer with the main CPU based on big.LITTLE architecture which integrates a quad-core ARM Cortex-A73 CPU cluster and a dual core Cortex-A53 cluster with a new generation Mali-G52 GPU.
  The large metal housing heatsink is designed to optimize the CPU and RAM heat dissipation and minimize throttling. The CPU is placed on the bottom side of the PCB to establish great thermal characteristics.
  It has 4 x USB 3.0, 1 x HDMI 2.0 and 1 x RJ45 Ethernet Port.
  It is powered by DC 12V/2A [8].

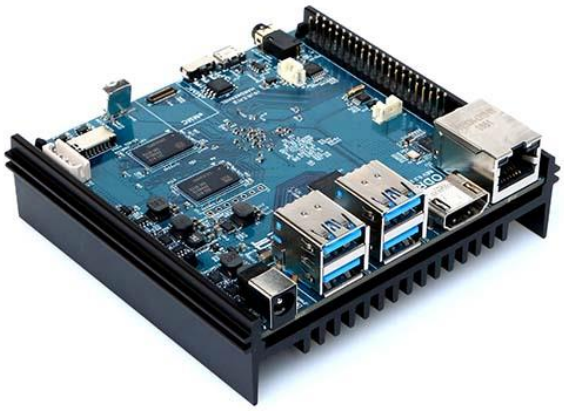


Figure 15: Odroid-N2

- **mvblueFOX3-M2004G**: it is a monochrome (G) compact industrial USB3 camera with a max frame rate of 436.9 Hz and a low resolution of 728x544 [9].
  The camera will be positioned close enough to the target (e.g. main rotor), so a low resolution is acceptable because, with this model, it has the advantage of having a fairly high frame rate.

Figure 16: MvblueFOX3-M2004G with optical lens

- **Spatial IMU**: it is a ruggedized miniature GPS aided inertial navigation system and AHRS that provides accurate position, velocity, acceleration and orientation under the most demanding conditions. It combines temperature calibrated accelerometers, gyroscopes, magnetometers and a pressure sensor with an advanced GNSS receiver. These are coupled in a sophisticated fusion algorithm to deliver accurate and reliable navigation and orientation [10].

This sensor can be connected to Odroid-N2 USB port and, thanks to its SDK, the programmer can directly access the data collected and use them as he prefers within his code.
The SDK can be downloaded directly from the Spatial website.
In this first application only the 3 accelerations along the X, Y and Z axes were used.

Figure 17: Spatial IMU

The PicoScope and the Brüel & Kjær accelerometers were not used because, due to the Covid-19, it was impossible to obtain this oscilloscope, consequently it was decided to continue developing the application with the instrumentation already available.

4.3 1st Application Implementation

First of all, it is necessary to download the OS image from the following link using a normal PC:
https://wiki.odroid.com/odroid-n2/os_images/ubuntu

Then, BalenaEtcher program is installed on a PC and it is used to install the operating system (Linux) on a SD card.
Then the SD card is connected to the Odroid-N2 and it is therefore possible to use Linux as an operating system on this device.

The camera driver is subsequently downloaded directly from the following website:
https://www.matrix-vision.com/serie-di-telecamere-industriali-compatte-usb3-vision-mvbluefox3-m2.html.

Finally, the OpenCV C++ library for Linux is downloaded: this is an open source C/C++ library for Image Processing and Computer Vision.

An Odroid-N2 has 6 available processing units which can be used to perform up to 6 operations in parallel. The **multithreading technique** is used to perform multiple operations in parallel in a C++ code: this technique consists in creating functions that are then performed in parallel. Each function is given as input to a different thread.
Each thread will be a **child process** executed inside the **parent process**, that is the *main* process: consequently, a *join* must be made for each thread so that the parent process is finished only when all the threads have ended. In fact, if the parent process was terminated before 1 or more child processes, an error would occur.

Code steps:

1. Camera acquisition and switching on;
2. Image acquisition: starting live loop;
3. Spatial IMU acquisition and switching on;
4. Create a viewing window;
5. Create acceleration graphs in the time domain;
6. Video and data saving;
7. Fast Fourier Transform;
8. End of software execution.

3 different threads are executed at the same time: these are created using the ***pthread_create()*** function:

```
// Initialize and set thread joinable
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);

pthread_create(&thread_id_geo, &attr, getGeoInfo, (void *)&GEO);
// create thread for getting IMU

pthread_create(&thread_id_blueFOX3, &attr, blueFOX3, (void *)&GEO);
// create thread to acquire blueFOX3 camera

pthread_create(&thread_id_save_video, &attr, save_video, (void
*)&VIDEO);
// create thread to save the video
```

As input, each thread has a thread *id*, a certain function to perform and a structure.

The 3 functions used are:

- **getGeoInfo**
  The first thread is the one related to the Spatial IMU sensor: the ***getGeoInfo*** function is used to activate the sensor and to detect the data of interest. In this case, accelerations along the 3 axes (X, Y, Z) are detected at each instant of time.

```
/* copy all the binary data into the typedef struct for the
packet */
/* this allows easy access to all the different values
*/
if(decode_raw_sensors_packet(&raw_sensors_packet, an_packet) ==
0)
{
      mtx_IMU.lock();
      GEO->IMU_SRC.acc_x = raw_sensors_packet.accelerometers[0];
      GEO->IMU_SRC.acc_y = raw_sensors_packet.accelerometers[1];
      GEO->IMU_SRC.acc_z = raw_sensors_packet.accelerometers[2];
      GEO->flag_acc = 1
      mtx_IMU.unlock();
}
```

In these lines of code, it is possible to note the use of the SDK to save the accelerations detected into a typedef struct called *GEO*: this operation allows easier access to the saved data.

- **blueFOX3**

  To turn on the camera and start acquiring images, it is first necessary to generate a setting file (*.xml* file) for setting the camera via **wxPropView**.
  To start wxPropView via the Linux terminal, just type the following command:
  wxPropView
  Once the program has been started and the desired setting parameters have been set, to generate the *.xml* file, go to *Action → Capture Settings → Save Active Device Settings → To a File*.

  Subsequently, a configuration file (*.conf*) is created in which to insert the serial number of the camera and the name of the setting file (*.xml* file).

```
img = Mat(pRequest->imageHeight.read(),
pRequest->imageWidth.read(), CV_8UC1,
pRequest->imageData.read(), pRequest->imageLinePitch.read() );

cvtColor(img, img_1, COLOR_BayerBG2GRAY);

mtx_blueFOX3.lock();
img_1.copyTo(GEO->blueFOX3);
mtx_blueFOX3.unlock();
```

The ***blueFOX3*** function allows the program to access the configuration file (*.conf*) and verify that the serial number of the camera is correct. Then it uses the ***liveLoop*** function to save the image in a local variable called *img*, transform it into a grayscale image called *img_1* and, finally, copy it into a global variable called *GEO.blueFox3.*

The mutex is used to prevent, when saving a data from a local variable to a global one, a memory location error due to the pointer.

For example, if a copy of a value, just saved within a vector in cell n°5, was made in another vector always in cell n°5 using the pointers, it can happen that the value saved in cell n°5 of the second vector actually is the value saved in cell n°6 or n°7 of the first vector.
That is, there may be a problem of overwriting the value that is to be saved when saving it to another vector.

In order to avoid having a similar problem between the thread of the *blueFOX3* function and the *SensorFusion*, two local variables and a global passing variable (*GEO.blueFOX3*) are used.

```
// get blueFOX3 final image
mtx_blueFOX3.lock();
GEO->blueFOX3.copyTo(canvas(ROI_final_img));
mtx_blueFOX3.unlock();
```

SensorFusion is a function that exactly allows to merge the various sensors, i.e. the blueFOX3 camera and the Spatial IMU: in fact, the data collected by these 2 instruments are used within this function.

Before analyzing the last thread related to saving the video, it is advisable to examine the work performed by the *SensorFusion* function.

This function was created to generate a window of suitable size in which to view both the image taken by the camera and 2D graphs representing the curves of the 3 accelerations in the time domain.
The graphs were created using the functions present in the **OpenCV** library: an appropriately sized vector, initially empty for each acceleration, was created. The vector is filled at each instant with the new measured value. Once the vector is completely filled at the *i-th* instant, it is transformed into a circular vector, i.e. the first element of the vector is eliminated, all its components are shifted by one cell to the left in order to free the last memory cell, and finally the last cell is filled with the acceleration value measured in that cycle.

The graphs were created using the following commands:
- ***arrowedLine***: for x (*time [s]*) and y (*acceleration* $[m^2/s]$) axes;
- ***line***: to join the various points representing the accelerations values measured at each instant of time.

First, the various matrices used for both camera image and graphics are initialized.

```
Mat canvas = Mat::zeros(1344,728,CV_8UC1);
Mat canvas_2 = Mat::zeros(728,1344,CV_8UC1);
...
// TIME DOMAIN window
Mat mat_x = Mat::zeros(200, 728, CV_64F);
Mat mat_y = Mat::zeros(200, 728, CV_64F);
Mat mat_z = Mat::zeros(200, 728, CV_64F);
Mat mat_time = Mat::zeros(100, 728, CV_64F);
```

Two initially empty matrices called *canvas* and *canvas_2* were created with equal size of the final window in order to be displayed. Then, using the ***Rect*** OpenCV function, rectangles are created within this matrix such as to reserve a certain number of pixels for each graph and for the image.

```
Rect ROI_final_img = Rect(0, 0, 728, 544);
Rect ROI_acc_x = Rect(0, 544, 728, 200);
Rect ROI_acc_y = Rect(0, 744, 728, 200);
Rect ROI_acc_z = Rect(0, 944, 728, 200);
Rect ROI_time = Rect(0, 1144, 728, 100);

Rect ROI_1 = Rect(1, 0, 727 - shift_y_axes, 1);
Rect ROI_2 = Rect(0, 0, 727 - shift_y_axes, 1);
```

ROI_1 and ROI_2 are used to create the circular vector.
ROI is the acronym of *Region Of Interest.*

Within an infinite *while* loop (*while(!flag)* with *flag=0*) all the operations necessary to display the final window are carried out.

Initially the cycle ends once the "*q*" or "*Q*" key is pressed ⇒ *flag=1.*

```
char c = (char)waitKey(10);

// Press q to exit from window
if( c == 27 || c == 'q' || c == 'Q' ) flag = 1;
```

At the beginning of the *while* loop the accelerations are copied from a global variable to a local one using the ***mutex*** (*lock* & *unlock*), as seen previously for the image.

```
// get IMU accelerations
mtx_IMU.lock();
acc_x = GEO->IMU_SRC.acc_x;
acc_y = GEO->IMU_SRC.acc_y;
acc_z = GEO->IMU_SRC.acc_z;
mtx_IMU.unlock();
```

Then the acceleration vectors begin to be filled. The *min* and *max* values of the accelerations present in each vector are calculated, normalized and finally scaled in order to be able to represent any value within the range of pixels reserved on the ordinate axis for each graph.
The previous functions are then used to create the various lines and, finally, to position the various graphs within the final display window.
The ***imshow*** command is used to show this final window to the user.

At the end of each iteration it is necessary to refresh the matrices containing the graphs to avoid incorrectly overwriting the pixels. To execute this refresh, each matrix has to be multiplied for 0 (black):

```
// Image refresh
mat_x = mat_x*0;
mat_y = mat_y*0;
mat_z = mat_z*0;
mat_time = mat_time*0;
```

At the bottom of the window a space has been used to insert the date and time calculated as follows:

```
time_t time_disp = time(nullptr);
TIME_str = ctime(&time_disp);

text = TIME_str.substr(0, TIME_str.length() -1);

putText(mat_time, text, Point(40, 40), FONT_HERSHEY_PLAIN, 1,
(255), 1, 1, false );
```

The ***putText*** function is used to insert a text string in a certain position within a pixel matrix.
Lines, texts and images are saved in a proper matrix which is finally copied into the respective ROI:

```
// x & y axes    &    accelerations lines
mat_x.copyTo(canvas(ROI_acc_x));
mat_y.copyTo(canvas(ROI_acc_y));
mat_z.copyTo(canvas(ROI_acc_z));
mat_time.copyTo(canvas(ROI_time));

// get blueFOX3 final image
mtx_blueFOX3.lock();
GEO->blueFOX3.copyTo(canvas(ROI_final_img));
mtx_blueFOX3.unlock();
```

The matrix representing the final window is called *canvas_2* and it is copied into a new struct called ***VIDEO***: that is, into a new global variable called *VIDEO.video_img*.
This struct is given as input to the last thread to save the video of the entire window, including the camera image, the graphs, the date and the time.

```
if(SAVING.input_status == 1)
{
      mtx_save_video.lock();
      canvas_2.copyTo(VIDEO->video_img);
      mtx_save_video.unlock();
}
```

- **save_video**
  In this last thread, always using the OpenCV functions, everything that appears in the final window is saved, so as to be able to view its content even in the *post-processing* phase.

```
VideoWriter video("./video.avi", CV_FOURCC('M', 'P', '4', 'V'),
15.0, img.size(), false);

while(VIDEO->exit_status==0)
{
      mtx_save_video.lock();
      VIDEO->video_img.copyTo(img);
      mtx_save_video.unlock();

      video.write(img);

}

video.release();
```

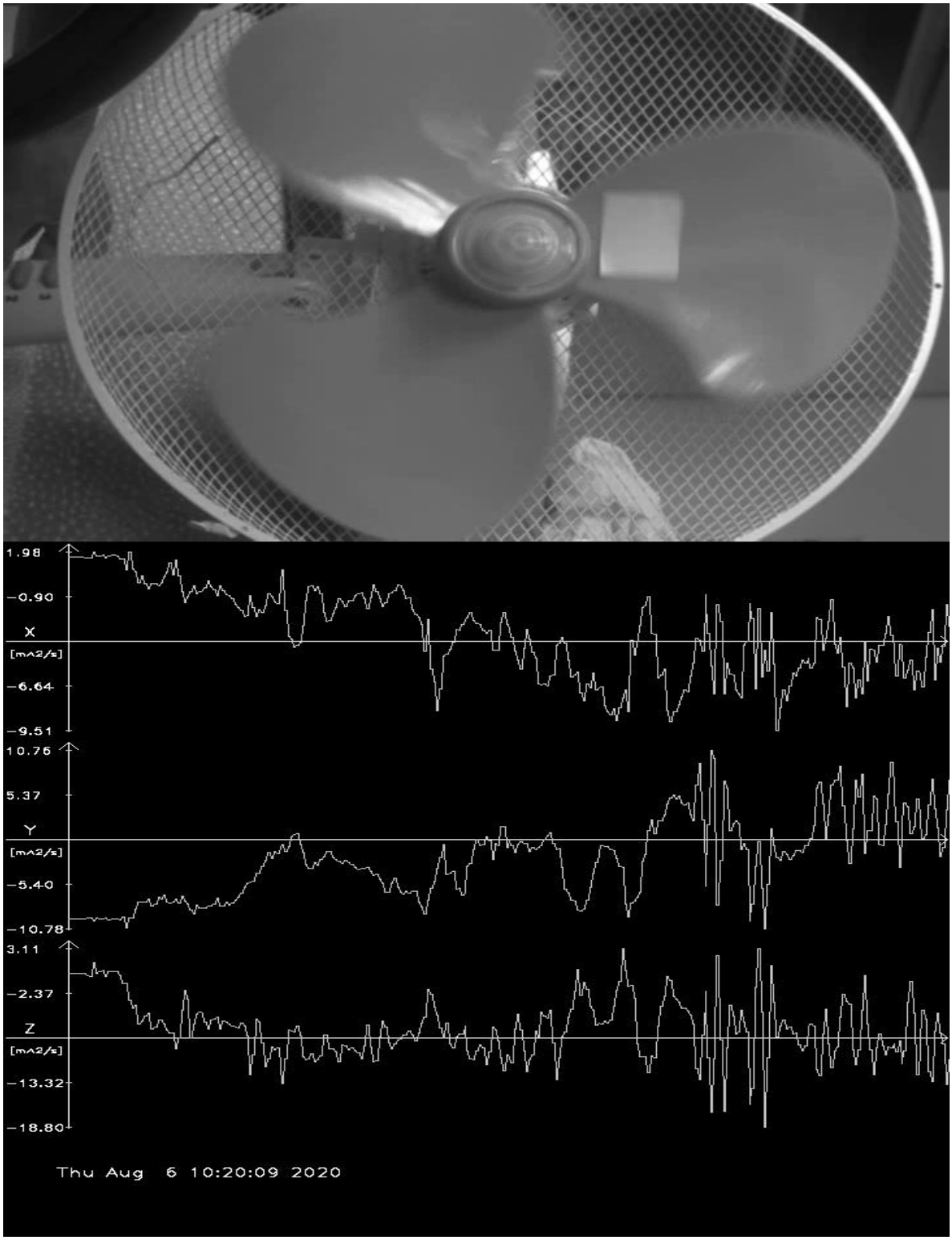


Figure 18: Software output window

The above figure is a screenshot of the final window seen by the user.

In this first version, some peculiarities can be noticed in the graphs: four values are written on the ordinate axes, i.e. the *max* and *min* values and two intermediate values. The origin of the graph does not coincide with the null value as otherwise, if for example there were only negative accelerations, only half of the space available for the graph would be used, making it more difficult for the user to understand the curve.

Looking at the window from top to bottom there are the following graphs:
- X axis accelerations;
- Y axis accelerations;
- Z axis accelerations;

All accelerations are calculated and represented in m^2/s.

After writing this first version of the code, a test was performed to verify that the images were captured at an appropriate frame rate.

A 3-blades domestic fan was therefore used as a test target. By sticking a post-it on a blade it is easier to identify its displacement in the lap.
In this way, it is possible to test the camera with something like a helicopter rotor.
A video was then recorded and then reviewed in slow motion in order to verify the movement of the blade during the lap.

4.4 Implementation of buttons on the screen

When a flight test is performed, it is necessary to save the data and the camera images only during some time windows of 10-20 seconds. For this reason, a code optimization could be the implementation of some **buttons** to start and finish saving data.
In this way there would also be a memory saving because only a small amount of data is saved.
In fact, the software is kept active throughout the flight test, but the data are saved only during these time windows.

So, initially three buttons are added:

- ***PLAY***: it allows to start saving data (accelerations) in a *.xls* file whose name contains the date and time when this button was pressed. Moreover, everything that appears on the screen from when the user presses the *PLAY* button until he decides to finish recording by pressing the *PAUSE* or *EXIT* button is saved in a video file. In this way also the camera images are saved.

Figure 19: PLAY button icon

- ***PAUSE***: it allows to pause saving data and camera images, but the software remains active and ready for a new time windows of data saving.

Figure 20: PAUSE button icon

- ***EXIT***: it allows to terminate the program execution. It works also as a final pause, i.e. if it is pressed during a data saving, it allows to terminate the program after saving all the data of the final time window.

Figure 21: EXIT button icon

A fourth button is added in order to perform a Fast Fourier Transform (FFT):

- ***FFT:*** it allows to execute an FFT of the last data recorded and to create another window with the acceleration graphs in the frequency domain.

Figure 22: FFT button icon

To let the user understand if the button was pressed correctly and/or which button was pressed the last time, a black square will appear inside the *PLAY* or *PAUSE* button icon.
The *EXIT* button replaces the "*q*" key with which the program was previously terminated.
These four buttons allow to provide the user a graphic interface with which to execute the various commands.

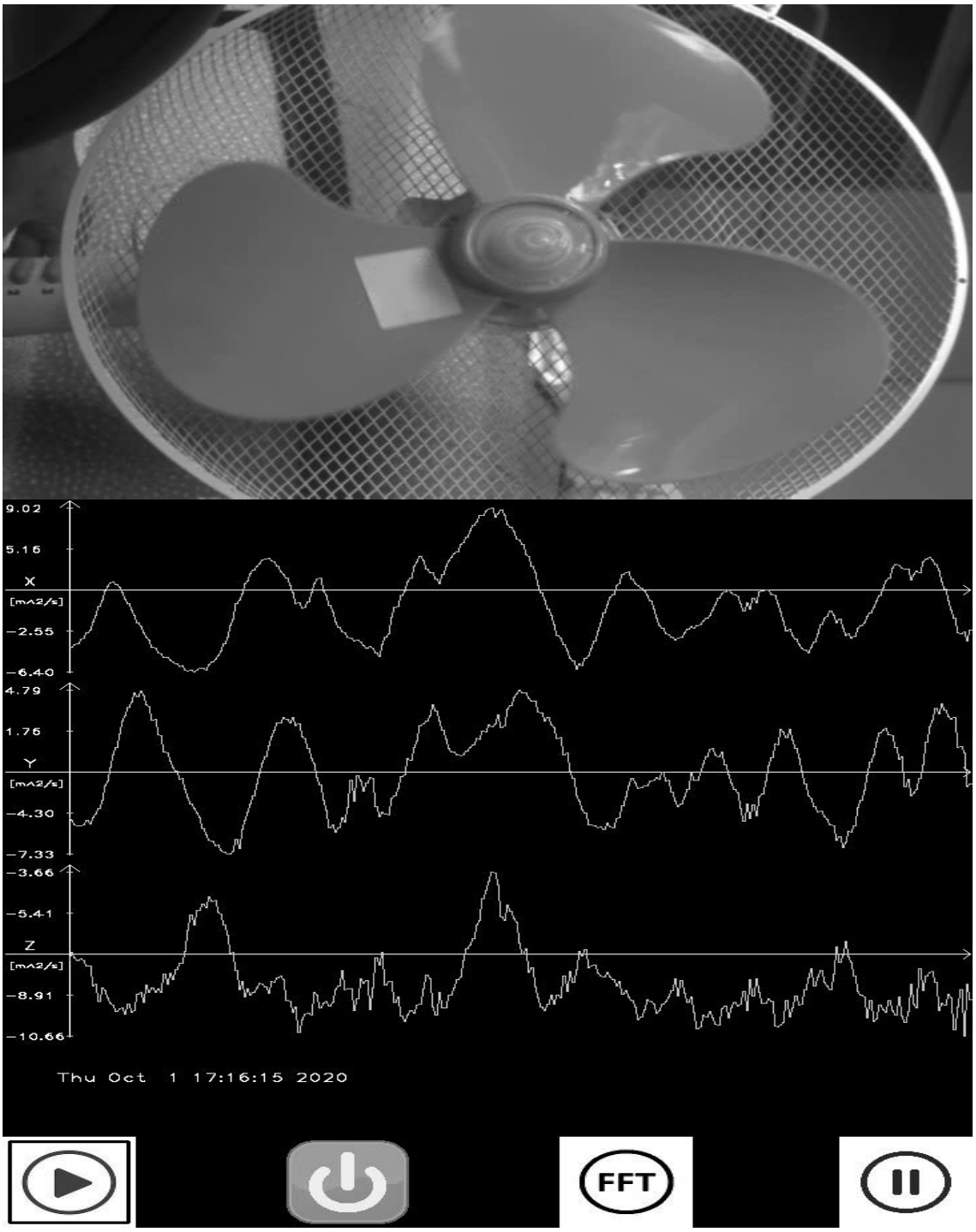


Figure 23: Time Domain Window

4.5 Fast Fourier Transform (FFT)

On the previous image there are four buttons: in fact, the last button added allows to perform a Fast Fourier Transform (FFT) of the last data recorded.

The Fast Fourier Transform (FFT) is an algorithm used to perform the Discrete Fourier Transform (DFT) or the Inverse Discrete Fourier Transform (IDFT) of a dataset. It is widely used not only in engineering and mathematical applications, but also in the musical and medical field (e.g. Magnetic Resonance Imaging (MRI), Computed Axial Tomography (CAT), etc.).

The Fourier Analysis allows to switch from the time domain to the frequency domain and vice versa: in this way different types of information usable in various field can be obtained, e.g. to verify that the frequencies detected are not too close to the resonance frequencies, in order to avoid a system collapse.

The DFT is a technique used to get the various frequencies associated with a series of vibration and/or acceleration values; unfortunately, it requires quite high computational time. Consequently, the FFT is often used as it is much quicker to perform as it has much shorter computational times.

The FFT is based on the factorization of the "*DFT matrix into a product of sparse factors, which are mostly zero".*
Considering *N* as data size, these two methods have the following computational costs:

- DFT → $O(N^2)$
- FFT → $O(N \cdot log_2(N))$

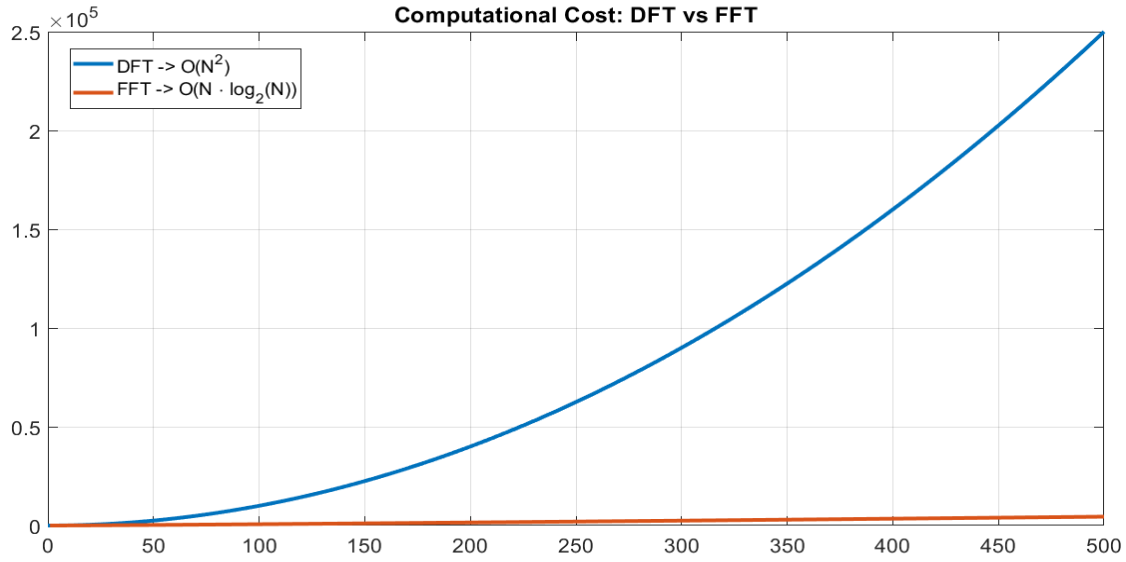


Figure 24: Computational Cost (DFT vs FFT)

Therefore, it is possible to notice that as *N* grows, the difference between these two computational costs grows more and more, until it becomes truly high if *N* is of the order of thousands or millions.

The Fourier Transform (FT) is a mathematic transformation performed on a certain *f* function so defined:

$$f: R^n \to C$$

Performing the FT of the *f* function, a new *F* function is obtained:

$$F(\xi) = (Ff)(\xi) = \frac{1}{(2\pi)^{\frac{n}{2}}} \int_{R^n} e^{-i\xi x} f(x) dx$$

whilst the Inverse Fourier Transform is:

$$F(-\xi) = \left(\tilde{F} f\right)(\xi) = \frac{1}{(2\pi)^{\frac{n}{2}}} \int e^{i\xi x} f(x) dx$$

with $\xi,\ x \in R^n$.
Considering a certain signal, it is possible to make a distinction in the case it is analyzed in the time domain or in the frequency domain.
In the time domain the signal is a *h* function which values are time dependent, so it is *h(t)*.
In the frequency domain, instead, the signal is characterized by a certain amplitude H (which is generally a complex number and it can have an initial phase) which depends on the frequency ν, so it is $H(\nu)$.
The Fourier Transform of a signal in the time domain allows to get its frequency distribution, i.e., considering the *h(t)* function, the following formulas are obtained:

$$H(\nu) = \int_{-\infty}^{+\infty} h(t) e^{2\pi i \nu t}\, dt$$
$$h(t) = \int_{-\infty}^{+\infty} H(\nu) e^{-2\pi i \nu t}\, dt$$

The Fourier Transform has some fundamental properties including the linearity property, i.e.:

- The Fourier Transform of a two functions sum is equal to the sum of the individual Fourier Transform.
- The Fourier Transform of a product between a *c* constant and a function is equal to the product of the *c* constant and the function Fourier Transform.

Moreover, if the *h(t)* function is an even or odd function, also its Fourier Transform will be an even or odd function [11].

4.6 Sampling Time and Sampling Rate: Signal Acquisition

When measuring an analog signal and converting it in a digital signal, a certain sampling time Δ must be considered. Δ is the time that elapses between one measurement and next, while the sampling rate $\nu_c = 1/\Delta$ is its reciprocal.

The samples of a time dependent signal *h(t)* are the following:

$$h_n = h(n\Delta)$$

where *n* is an integer.

The sampling rate must be chosen considering the **Nyquist-Shannon Theorem**, according to which the sampling rate ν_c must be at least two times greater than the maximum rate ν_{max} to be detected, so:

$$\nu_c \geq 2 \cdot \nu_{max}$$

The half of the sampling rate is called **Nyquist critical frequency**:

$$\nu_n = \frac{\nu_c}{2} = \frac{1}{2\Delta}$$

The Nyquist-Shannon theorem considers the **aliasing problem**, that occurs in the moment which the maximum frequency ν_{max} is greater than ν_n. In that case, the $H(\nu)$ values that should be out of the $[-\nu_n, \nu_n]$ range are translated inside this range causing a signal distortion.

To avoid the aliasing problem, some low pass filter or a greater sampling rate are used. Usually the maximum sampling rate depends on the device used, but, if it is unknown, it is possible to understand if there is an aliasing problem analyzing the Fourier Transform behavior when it is near the Nyquist frequency range limits:

- If the Fourier Transform is almost 0 near these limits, the aliasing problem is minimized.
- If the Fourier Transform stabilizes on a constant value different from 0, the aliasing effects are not negligible, so there is a signal distortion.

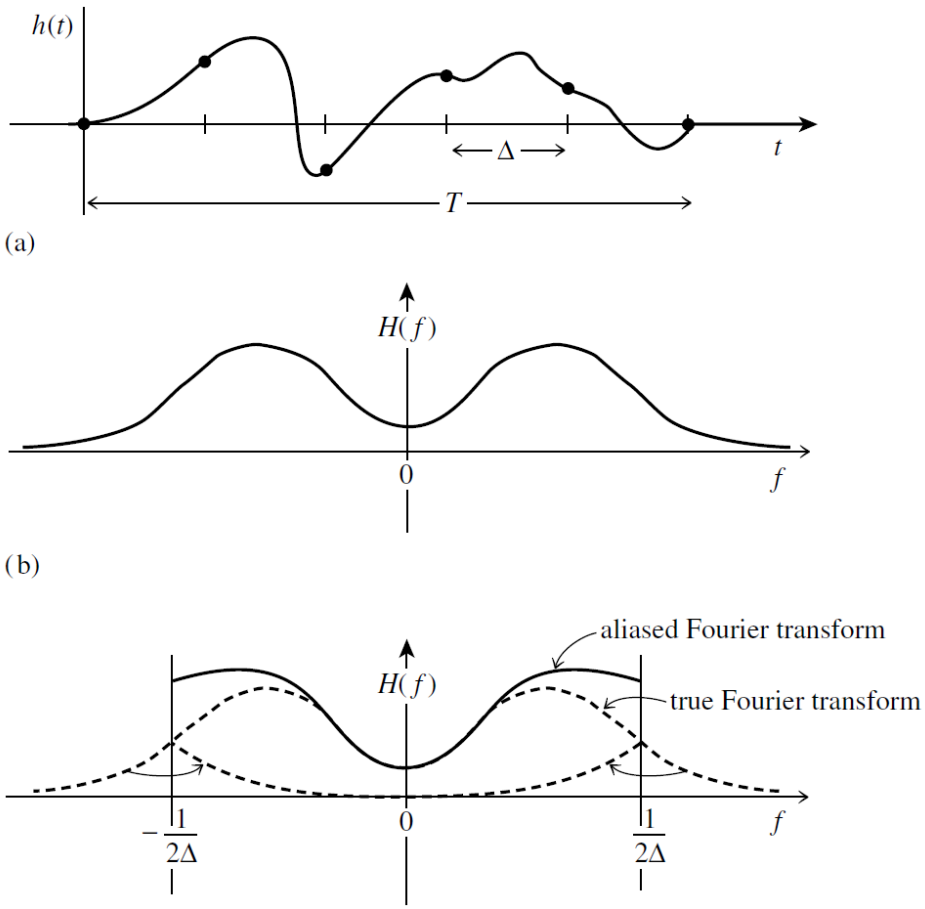


Figure 25: Aliasing Problem [11]

4.7 Fourier Transform Discretization

The Fourier Transform is a continue function, but to execute a Discrete Fourier Transform is necessary to discretize it. By sampling the input signal, *N* consecutive samples are obtained:

$$h_k = h(t_k) \qquad \text{with} \qquad t_k = k \qquad \text{and} \qquad k = 0,\ 1,\ 2,\ \ldots,\ N-1$$

Having *N* input samples there will be *N* output samples, so it is possible to consider only the discrete frequency value:

$$\nu_n = \frac{n}{N\Delta} \qquad \text{with} \qquad n = -\frac{N}{2},\ -\frac{N}{3},\ \ldots,\ +\frac{N}{2}$$

The lower and upper limits correspond to the Nyquist critical frequency, so they are not independent like the other frequencies. Consequently, only *N* frequencies are considered and therefore it is possible to approximate the Fourier Transform in this way:

$$H(\nu_n) = \int_{-\infty}^{+\infty} h(t) e^{2\pi i \nu_n t}\, dt \simeq \sum_{k=0}^{N-1} h_k\, e^{2\pi i \nu_n t_k} \Delta = \Delta \sum_{k=0}^{N-1} h_k\, e^{2\pi i k n/N}$$

The **Discrete Fourier Transform** (**DFT**) of *N*-h_k samples is:

$$H_n = \sum_{k=0}^{N-1} h_k e^{2\pi i k n/N}$$

Therefore:

$$H(\nu_n) \simeq \Delta H_n$$

The Inverse Discrete Fourier Transform is:

$$h_k = \frac{1}{N} \sum_{n=0}^{N-1} H_n e^{-2\pi i k n/N}$$

4.8 Cooley-Tukey FFT algorithm

To perform a DFT it is necessary to perform some operations: it is possible to rewrite the previous equation of DFT considering *N* samples and a complex number W_N:

$$W_N = e^{2\pi i/N}$$

Consequently:

$$H_n = \sum_{k=0}^{N-1} h_k\, W^{kn}$$

W^{kn} is a complex matrix of dimensions *N* x *N*.

To obtain H_n, N^2 operations must be performed. There are some algorithms that allow to decrease significantly the number of operations, as the **Cooley-Tukey factorization algorithm**, which is based on the idea of simplifying the calculations to be made by decomposing the problem into simpler and faster subproblems to solve.
Thanks to this algorithm, only $Nlog_2(N)$ operations must be performed for an FFT, consequently the computational time is much lower than that of the DFT.

Moreover, thanks to the **Danielson-Lanczos lemma** it is possible "*to rewrite a DFT of length N as the sum of two DFTs, each of length N/2. One is formed from the even-numbered points ("e" apex), while the other from the odd-numbered points ("o" apex)*" [12].

Therefore:

$$H_n = \sum_{j=0}^{N-1} h_j \, e^{\frac{2\pi i j n}{N}} = \sum_{j=0}^{N/2-1} h_{2j} \, e^{\frac{2\pi i (2j) n}{N}} + \sum_{j=0}^{N/2-1} h_{2j+1} \, e^{\frac{2\pi i (2j+1) n}{N}} =$$

$$= \sum_{j=0}^{N/2-1} h_{2j} \, e^{\frac{2\pi i j n}{N/2}} + \sum_{j=0}^{N/2-1} h_{2j+1} \, e^{\frac{2\pi i j n}{N/2}} \cdot e^{\frac{2\pi i n}{N}}$$

$$= \sum_{j=0}^{N/2-1} h_{2j} \, e^{\frac{2\pi i j n}{N/2}} + W_N^n \sum_{j=0}^{N/2-1} h_{2j+1} \, e^{\frac{2\pi i j n}{N/2}}$$

$$\Rightarrow \quad H_n = F_n^e + W_N^n F_n^o$$

The only problem of this algorithm is that it is valid only for a *N* samples that is power of 2. In fact, otherwise, it would be impossible to find a number of operations equal to $log_2(N)$, consequently the method would not be valid. However, there is a solution for these cases: it is possible to fill the data pattern with null terms until the next power of 2.
In this way the signal is not altered because the null terms inserted do not make any contribution to the Fast Fourier Transform (FFT).

Finally, this algorithm is implemented in a C++ code in order to perform an FFT of the last dataset saved when the user presses the *FFT* button.
If the user presses the *FFT* button, a new window is opened where the graphs obtained in the frequency domain are displayed.

Assume to have the following sinusoidal input signals:

- X-axes: $sin(16 \cdot \pi \cdot t) + cos(40 \cdot \pi \cdot t)$
- Y-axes: $sin(8 \cdot \pi \cdot t) + cos(24 \cdot \pi \cdot t)$
- Z-axes: $sin(12 \cdot \pi \cdot t) + cos(50 \cdot \pi \cdot t)$

Considering these input signals, the frequencies expected to find in the frequency domain are the following:

- X-axes: $f_1 = 8\ Hz$ and $f_2 = 20\ Hz$
- Y-axes: $f_1 = 4\ Hz$ and $f_2 = 12\ Hz$
- Z-axes: $f_1 = 6\ Hz$ and $f_2 = 25\ Hz$

Pressing the *FFT* button the FFT is performed, so the frequency domain window shown is the following:

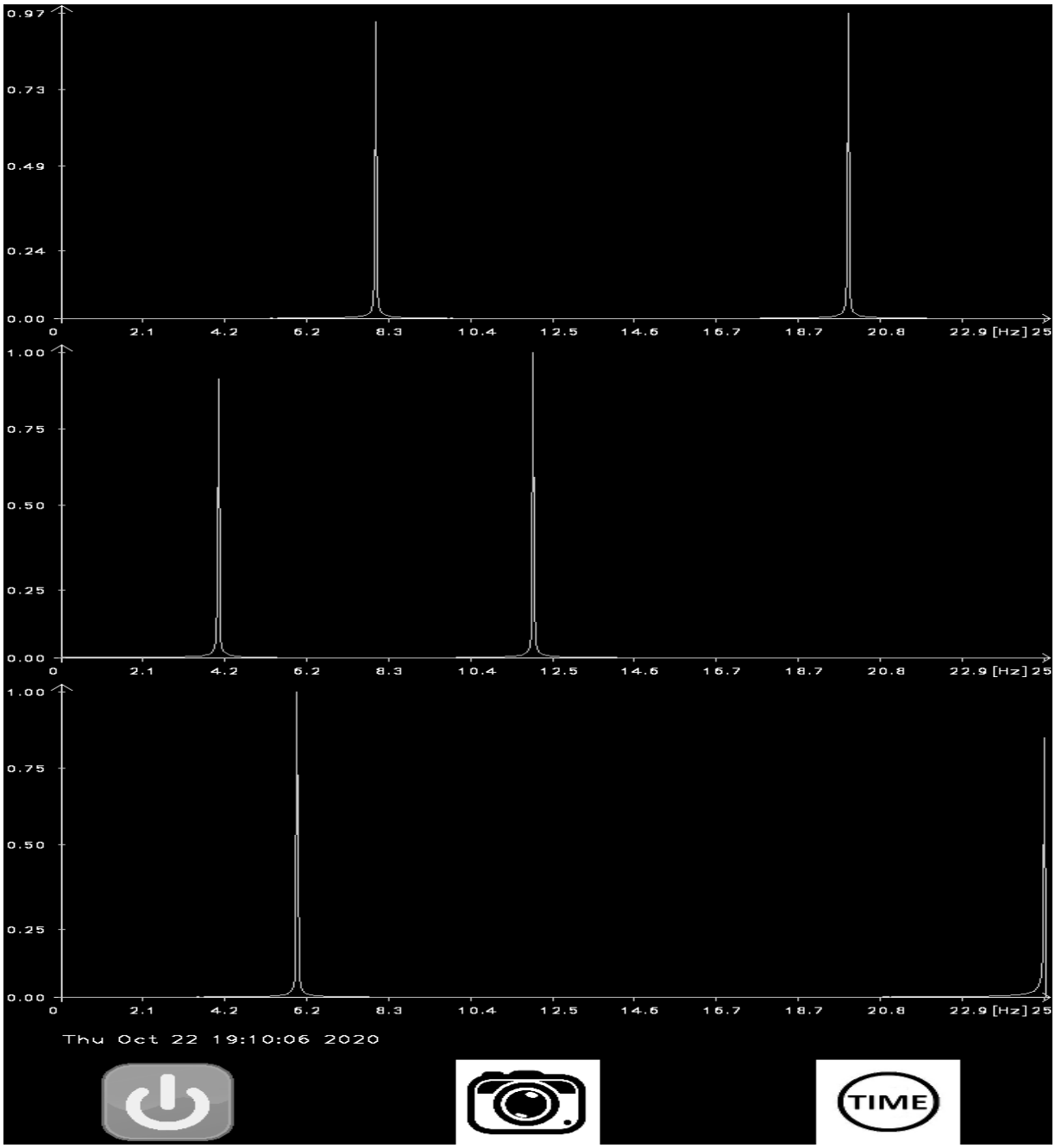


Figure 26: Frequency Domain Window - FFT ideal case

Such a clean graph can be obtained in an **ideal case** where there is not an error range of the sensor used and where there is only a pure sinusoidal signal.

A graph obtained in a **real case**, on the other hand, can be the following, in which the dominant frequencies (i.e. the peaks) are visible, but in which there are very small values for almost all frequencies.

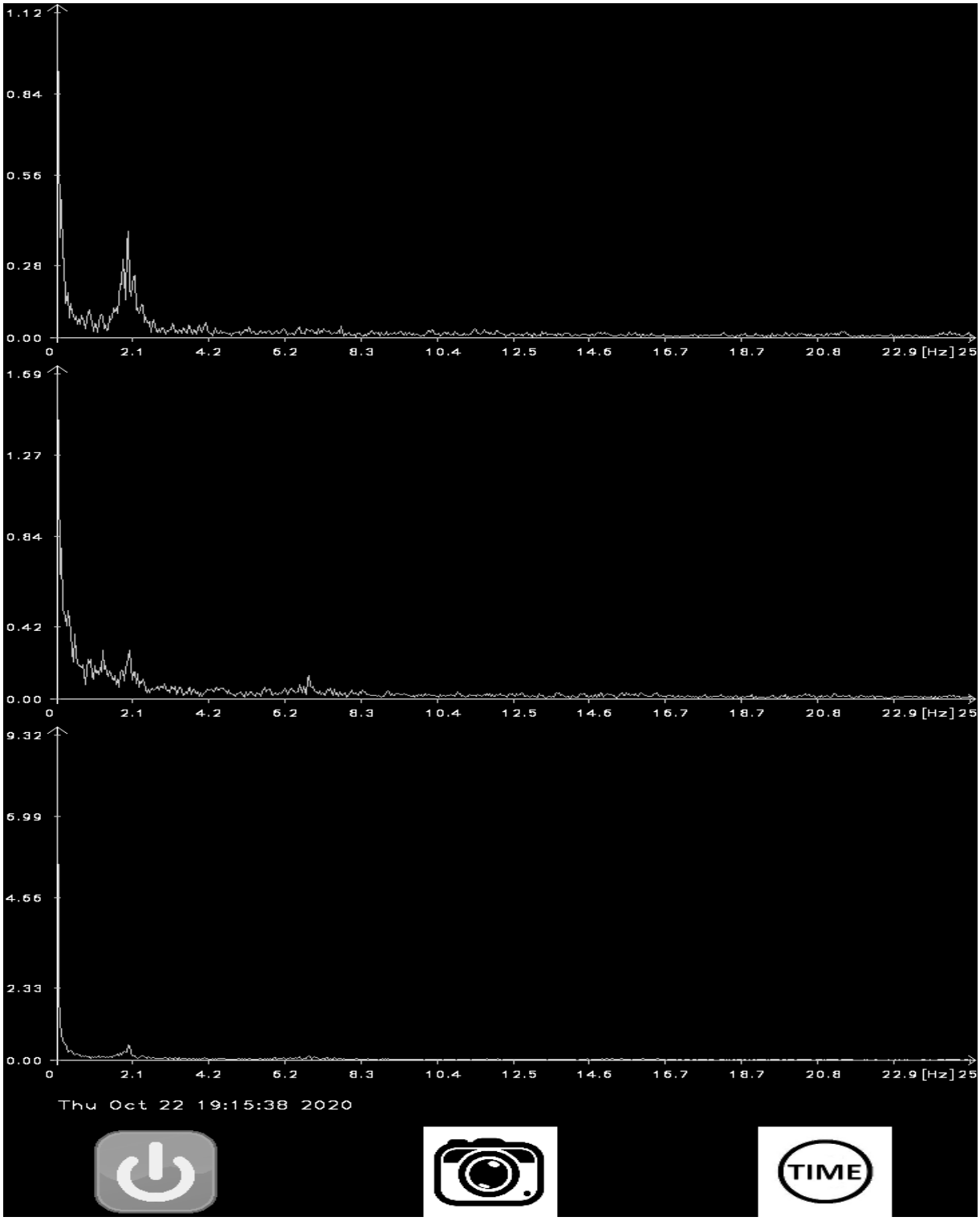


Figure 27: Frequency Domain Window - FFT real case

Each sensor, in fact, has a certain wide error range that can affect the data detected.

In this window there are two new buttons:

- ***CAMERA button***: it allows to do a screenshot of the frequency domain window, in order to allow the user to view even during the flight test the FFT performed of all the data recorded.

Figure 28: CAMERA button icon

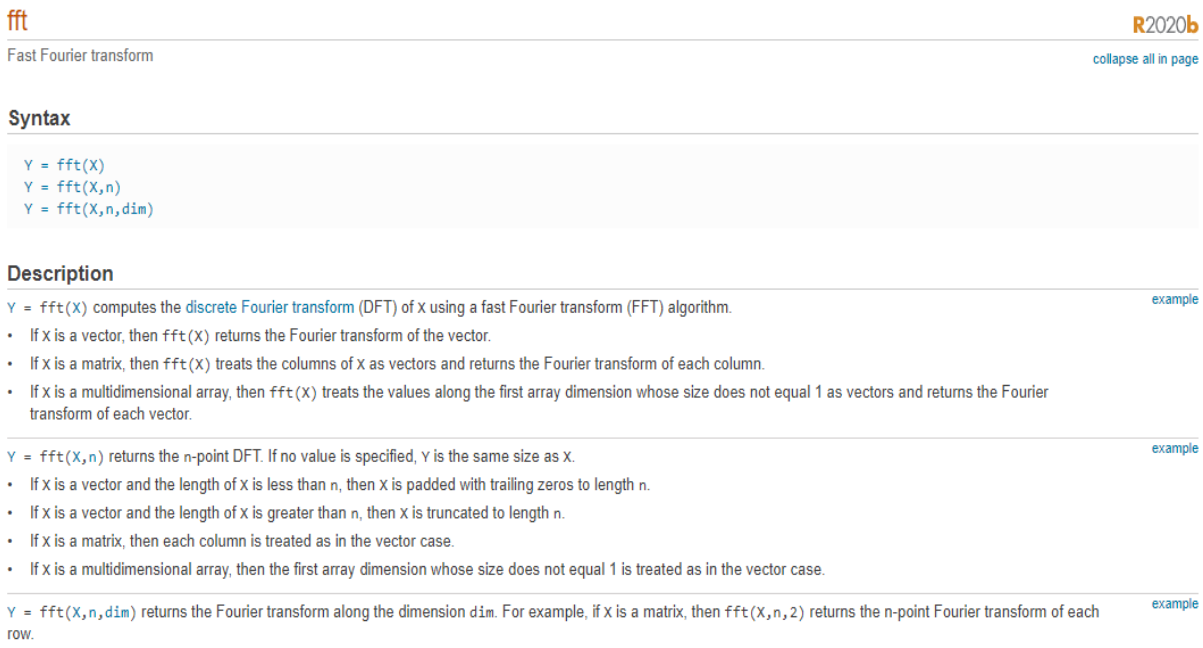
- ***TIME button***: it allows to return to the time domain window in order to start a new data recording. When the *PLAY* or *PAUSE* button is pressed, a new dataset is recorded, so it is impossible to view the FFT of the previous dataset. For this reason, the *CAMERA* button is added.

Figure 29: TIME button icon

However, the data obtained thanks the FFT are saved on a text file, thus the user can use it during the post-processing phase.

4.9 FFT C++ code validation

To verify if the C++ code that performs the Fast Fourier Transform is correct, a comparison was made with a MATLAB code using the *fft* function implemented in this software.



fft

R2020b

Fast Fourier transform

collapse all in page

Syntax

```
Y = fft(X)
Y = fft(X,n)
Y = fft(X,n,dim)
```

Description

`Y = fft(X)` computes the discrete Fourier transform (DFT) of `X` using a fast Fourier transform (FFT) algorithm. example

- If `X` is a vector, then `fft(X)` returns the Fourier transform of the vector.
- If `X` is a matrix, then `fft(X)` treats the columns of `X` as vectors and returns the Fourier transform of each column.
- If `X` is a multidimensional array, then `fft(X)` treats the values along the first array dimension whose size does not equal 1 as vectors and returns the Fourier transform of each vector.

`Y = fft(X,n)` returns the `n`-point DFT. If no value is specified, `Y` is the same size as `X`. example

- If `X` is a vector and the length of `X` is less than `n`, then `X` is padded with trailing zeros to length `n`.
- If `X` is a vector and the length of `X` is greater than `n`, then `X` is truncated to length `n`.
- If `X` is a matrix, then each column is treated as in the vector case.
- If `X` is a multidimensional array, then the first array dimension whose size does not equal 1 is treated as in the vector case.

`Y = fft(X,n,dim)` returns the Fourier transform along the dimension `dim`. For example, if `X` is a matrix, then `fft(X,n,2)` returns the n-point Fourier transform of each row. example

Figure 30: fft function MATLAB Documentation [13]

This is the MATLAB Documentation about the *fft* function.

The validation test is performed considering 3 sinusoidal input signals with different frequencies:

- 1st input signal: $sin(10 \cdot \pi \cdot t)$
- 2nd input signal: $sin(20 \cdot \pi \cdot t)$
- 3rd input signal: $sin(30 \cdot \pi \cdot t)$

In the C++ code 3 accelerations along each axis (X, Y and Z) are used as input signals, therefore the above written signals are used as input signals respectively for X, Y and Z axis.

The following results are obtained with the MATLAB and C++ codes:

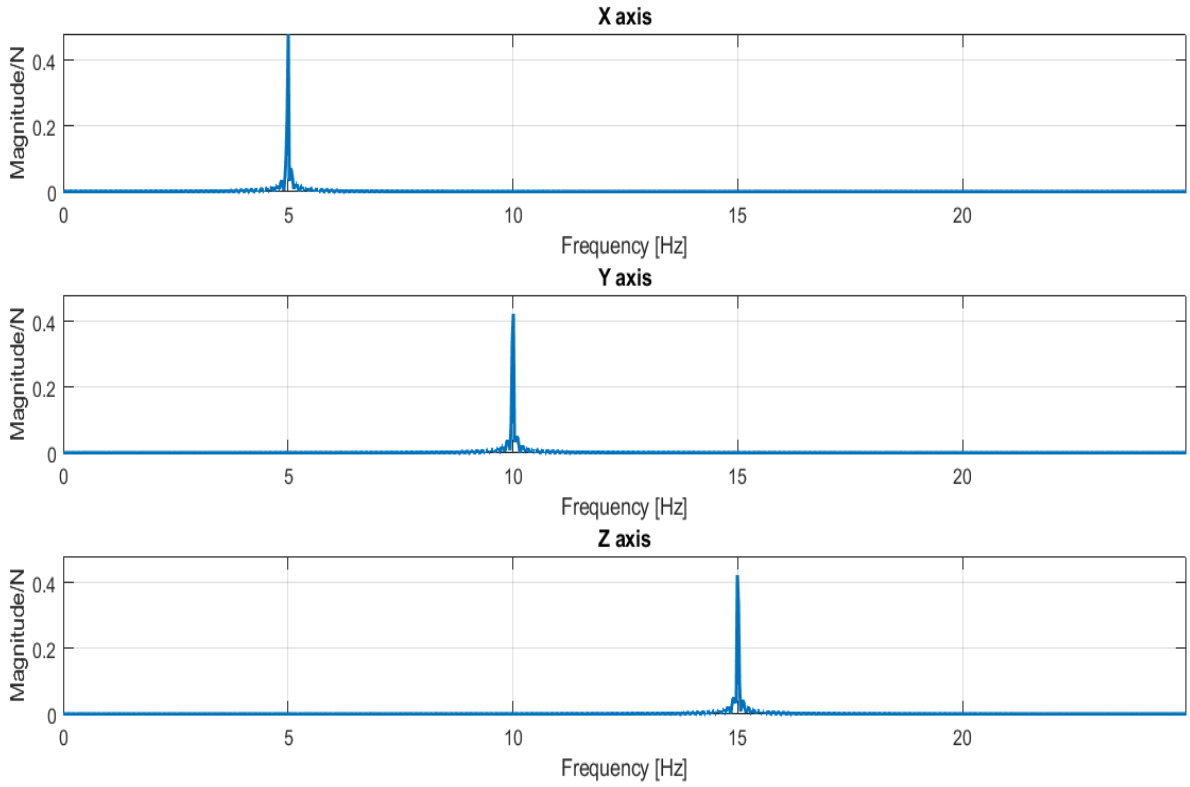


Figure 31: MATLAB results

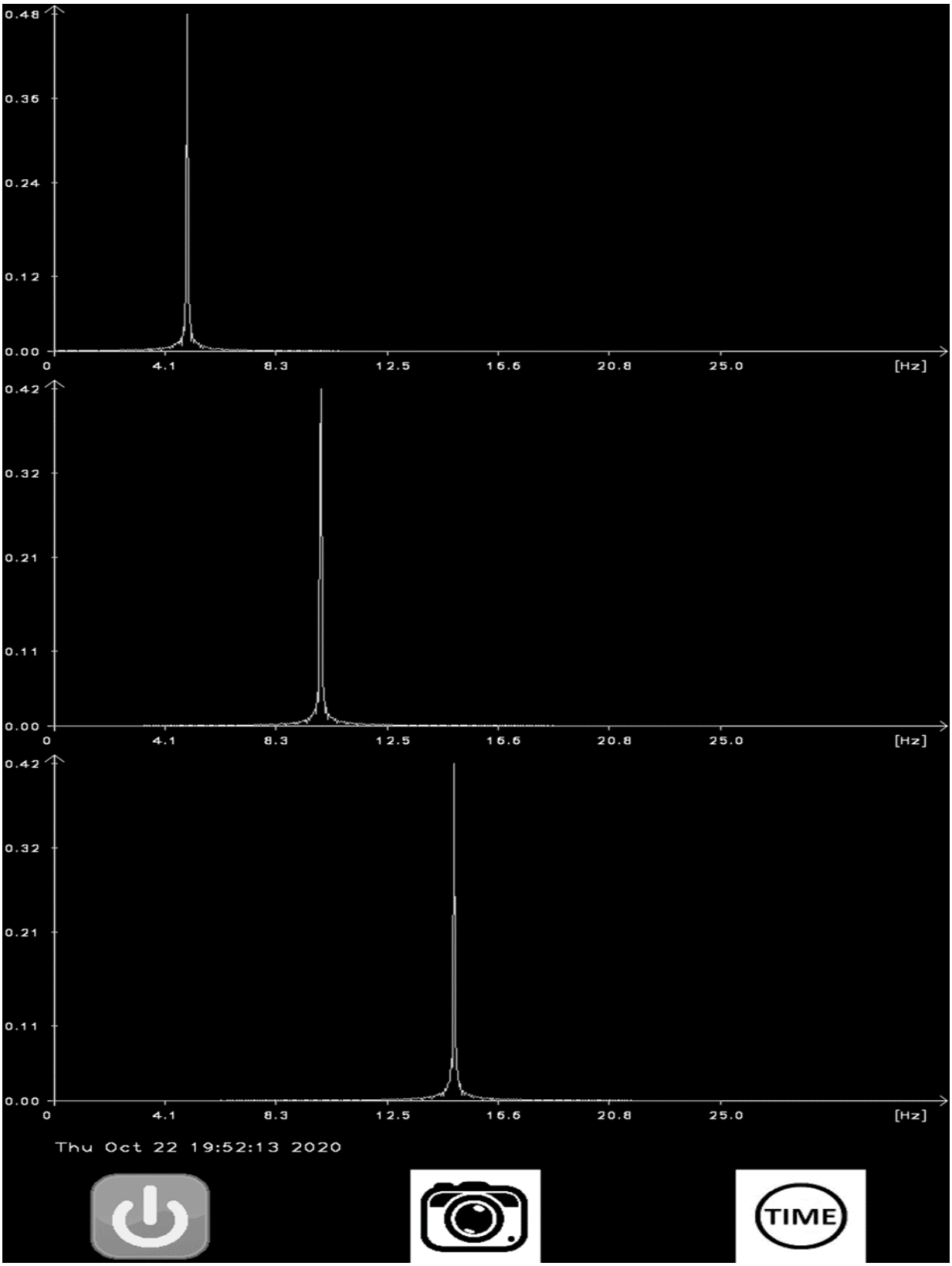


Figure 32: C++ code results

On the abscissa axes there is the frequency in Hertz, while on the ordinate axes there is the ratio between the magnitude and the number of samples *N* saved. In this case *N* = 1602, so for the FFT the next power of 2 closest to *N* used is *n* = 2048.

It is important to notice that both *x* and *y* values are the same on the figures obtained from the two different codes, therefore, the *fft* MATLAB function and the C++ code implemented work the same way.

So, why write a C++ code that performs the Fast Fourier Transform when instead it is possible to use directly the fft function on MATLAB?

There are several reasons: first, using a C++ code it is not necessary to have a MATLAB license and to download MATLAB on the computer. Therefore, the user can also save money from an economic point of view.
Moreover, in order to use MATLAB, the user should use multiple programs (C++ code to acquire data and MATLAB to perform the FFT), consequently the use of the various programs would be made more difficult and less immediate.
The MATLAB code would also require the opening of files, saving the data and creating the various figures representing the graphs. This whole process would require a greater amount of time than the time required to press a button that allows to perform all this instantly. In this way, the user can save a lot of time and simplify his work a lot.

However, in the C++ code, thanks to the text file where the data obtained with the FFT are saved, the user can also use them during the post-processing phase for other purposes.

4.10 1st Application Conclusions

Once the Fast Fourier Transform is performed, the user can view the data in the frequency domain, and he can compare them to the previous data or to the accelerations prescribed by the Regulations.

This application allows the Flight Test Engineer to use a non-invasive instrumentation, easy to transport and to mount on the aircraft, and to carry out the various analyses in a simpler and more immediate way.

The Flight Test Engineer can then perform analysis in real time when he is still on board the aircraft: in this way, if from an analysis there are non-compliant results, it is possible to repeat the maneuver immediately to try to understand, for example, why there is a certain vibration. With the previous instrumentation, however, the Flight Test Engineer could carry out these analyses only once landed, consequently it would

have been necessary to make the aircraft take off again, with a consequent loss of time and an increase in flight test costs.

5 Parameter Identification

The Parameter Identification is a technique born around the 70s and concerns the analysis of the **control** and **stability characteristics** of aircraft.

What is the difference between stability and control derivatives?

- **Stability derivatives**: measure how much forces and/or moments acting on the aircraft change as a result of a small change in flight condition parameter such as airspeed, altitude, angle of attack, etc.

- **Control derivatives**: measure how much forces and/or moments acting on the aircraft change as a result of a small change in the deflection of a control surface as rudder, elevator and aileron [13].

"Parameter Identification has become a significant tool for applications such as model validation, handling qualities evaluation, control law design, and flight-vehicle design and certification" [14].

Aircraft in flight have a dynamic like that of a mass-spring-damper system (II order system) but they have at least 6 degrees of freedom (DOF): 3 translational and 3 rotational (roll, pitch, yaw).
For a helicopter 9 DOF are considered because there are also 3 DOF of the blade: lead-lag, flap and pitch.

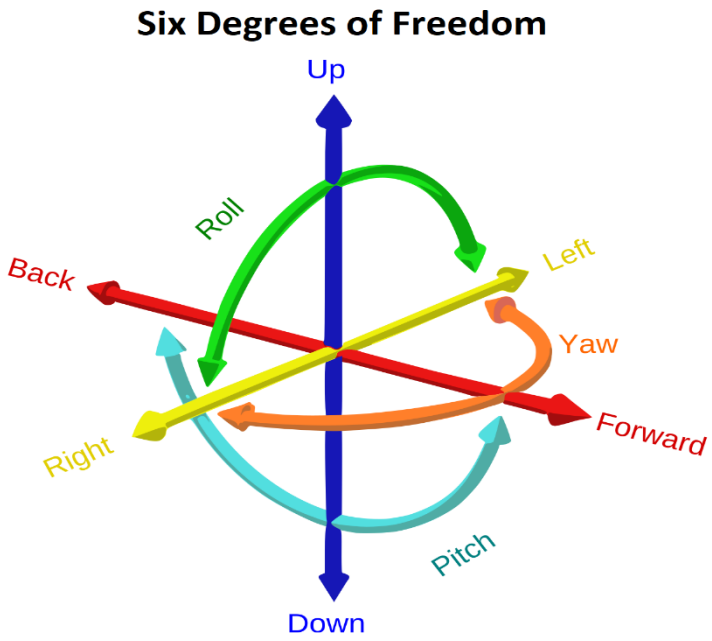


Figure 33: Six Degrees of Freedom [15]

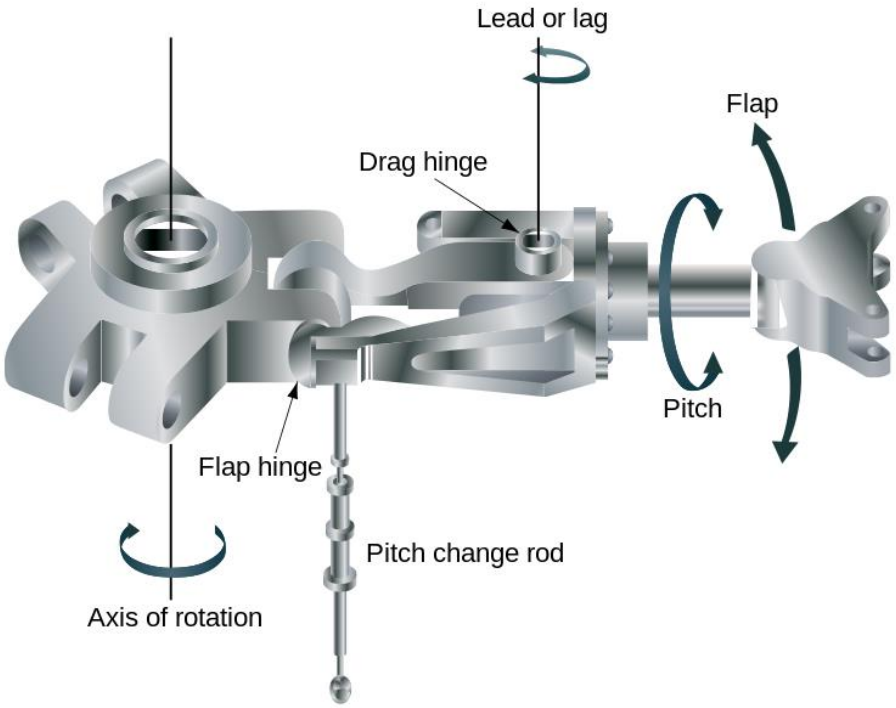


Figure 34: Helicopter rotor blades degrees of freedom [16]

For the aircraft dynamic analysis 6 linearized equations of motion are usually used, within which there are some terms called *stability* and *control derivatives*.
The value of many of these parameters depends on the speed, in fact they can be quite different between the hover condition and the forward flight condition.

Usually a first estimate of these values is obtained even before the first flight of the aircraft. This estimation is made through a combination of analysis, wind-tunnel tests and assumptions made with some judgment.

Unfortunately, however, when the aircraft is in flight, it could have a different behavior than the one predicted by these initial estimates. For this reason, flight-tests are carried out with which to obtain more precise information relating to the values of the stability and control derivatives.

In fact, the data obtained from the wind-tunnel tests (*computed*) are very different from those obtained during flight tests (*flight*), as shown in the following figure:

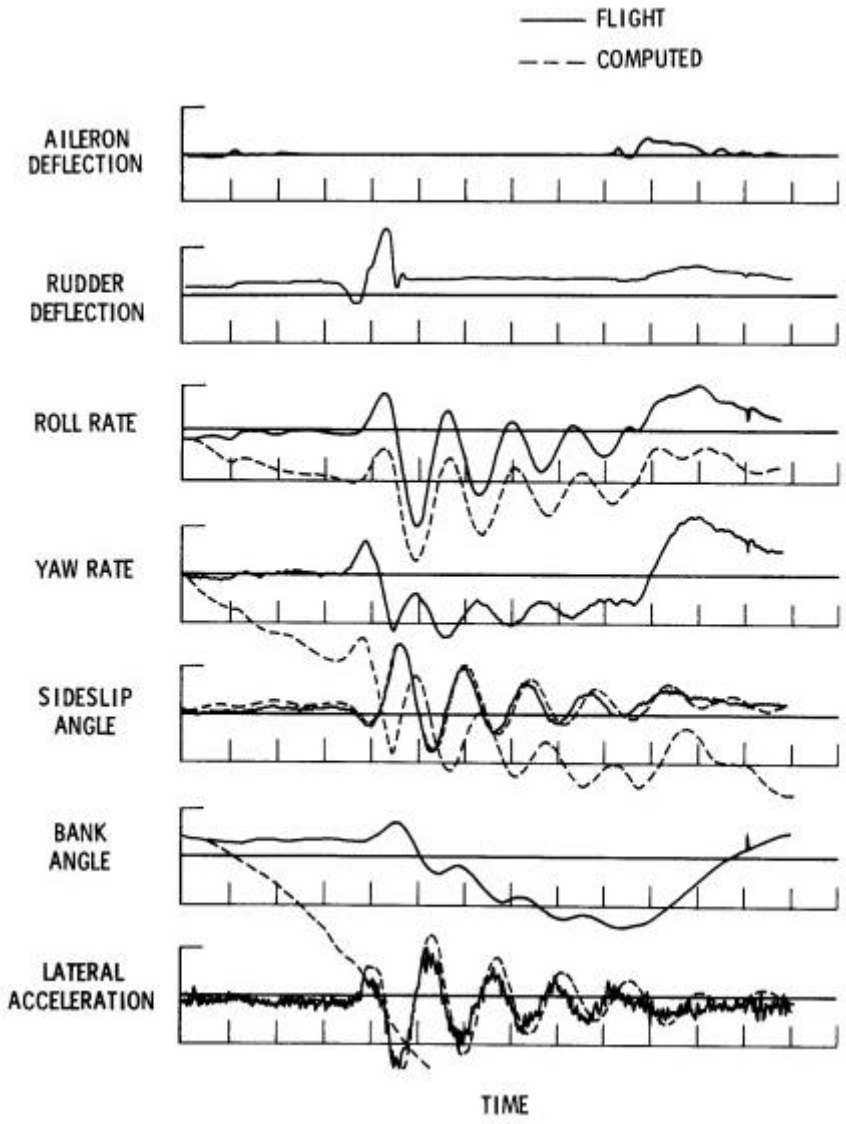


Figure 35: Comparison of computed response using wind-tunnel-parameter values with flight-measured response [17]

The results obtained through the flight-tests are very important for a subsequent development of flight simulators of a specific *helicopter or category of helicopters*, but also to carry out a better analysis to correct flying-quality problems or to develop autopilots.

Using flight-test data allow to eliminate the errors generated by the assumptions and approximations made during the initial estimates.

Before performing a flight-test campaign, it's very important to choose what maneuvers realize modeling the commands input. This choice is fundamental because it allows to minimize the uncertainties present in the Parameter Estimation procedure and to maximize the flight-test data content.
To perform this optimization in the command input modeling it is necessary to have a priori knowledge about the dynamic of that specific helicopter [18].

How are flight-tests performed?
Before carrying out a flight test, the aircraft is instrumented with appropriate sensors capable of measuring certain parameters: for example, accelerations, speeds and the attitude that the aircraft assumes during flight.
Starting from a trim condition, the pilot performs one or more maneuvers using the various commands: in the case of a helicopter the pilot uses all four pilot control inputs: collective, longitudinal cyclic, lateral cyclic and pedals.
There are different types of standard input commands, e.g. step, pulse, doublet or "3211".
The "3211" is a sequence of sharp-edged pulses that excite both the short-period and long-period (phugoid) modes and each number corresponds to each input duration [19].

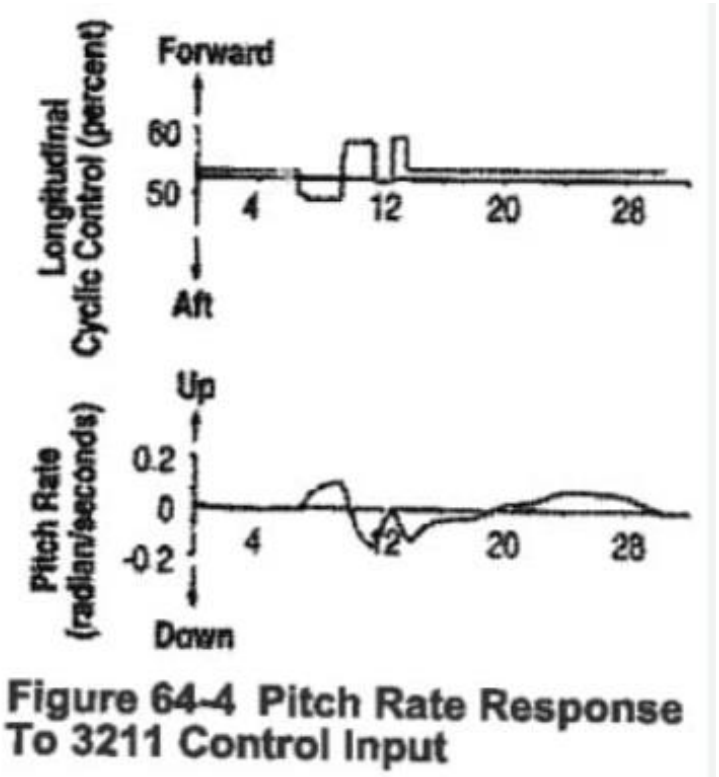


Figure 36: 3211 control input [5]

Usually these tests are carried out with the Stability and Control Augmentation System (SCAS) turned off, except in the case the helicopter is so unstable that it does not allow to obtain adequate time histories of the various parameters.

Some of the data that must be measured during a flight-test are fuel quantity in each tank, nose boom static and dynamic pressures, external stagnation temperature, aerodynamic angle of attack (***α***) and sideslip angle (**β**), roll, pitch, and yaw rates (***p***, ***q***, and ***r***, resp.) and accelerations, body axes speeds (***u***, ***v***, ***w***) and accelerations, load factors, longitudinal (**θ**) and lateral (**ϕ**) body attitudes, heading, collective, longitudinal and lateral cyclic, and pedal command deflections ($\boldsymbol{\delta_C}$, $\boldsymbol{\delta_B}$, $\boldsymbol{\delta_A}$ and $\boldsymbol{\delta_P}$, resp.) [18].

Therefore, both input and output data of the real system are measured.
To measure the inputs given by the pilot, it is also necessary to instrument the flight controls with appropriate sensors that allow to have some information about commands amplitude and duration.

When carrying out flight tests, the data detected by the sensors may be subject to noise caused by the vibrations of the sensors' cables, therefore, usually, a Kalman Filter is used to mitigate the influence of noise on the data measured. In fact, using data affected by noise would substantially influence the final analysis. Moreover, there may be also external disturbances that are impossible to measure directly.

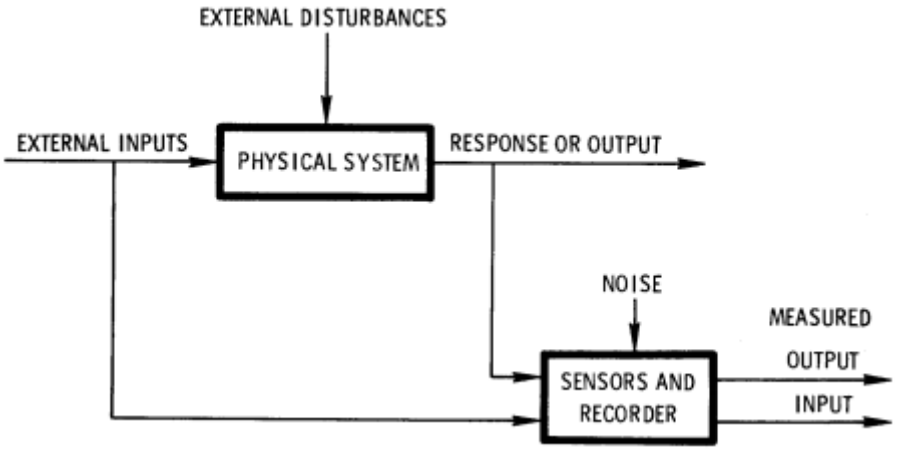


Figure 37: General systems identification problems [17]

If during a flight-test aileron and rudder small-amplitude pulses are performed, the measured output data are roll and yaw rate, sideslip and bank angle, lateral acceleration, as shown in the following figure:

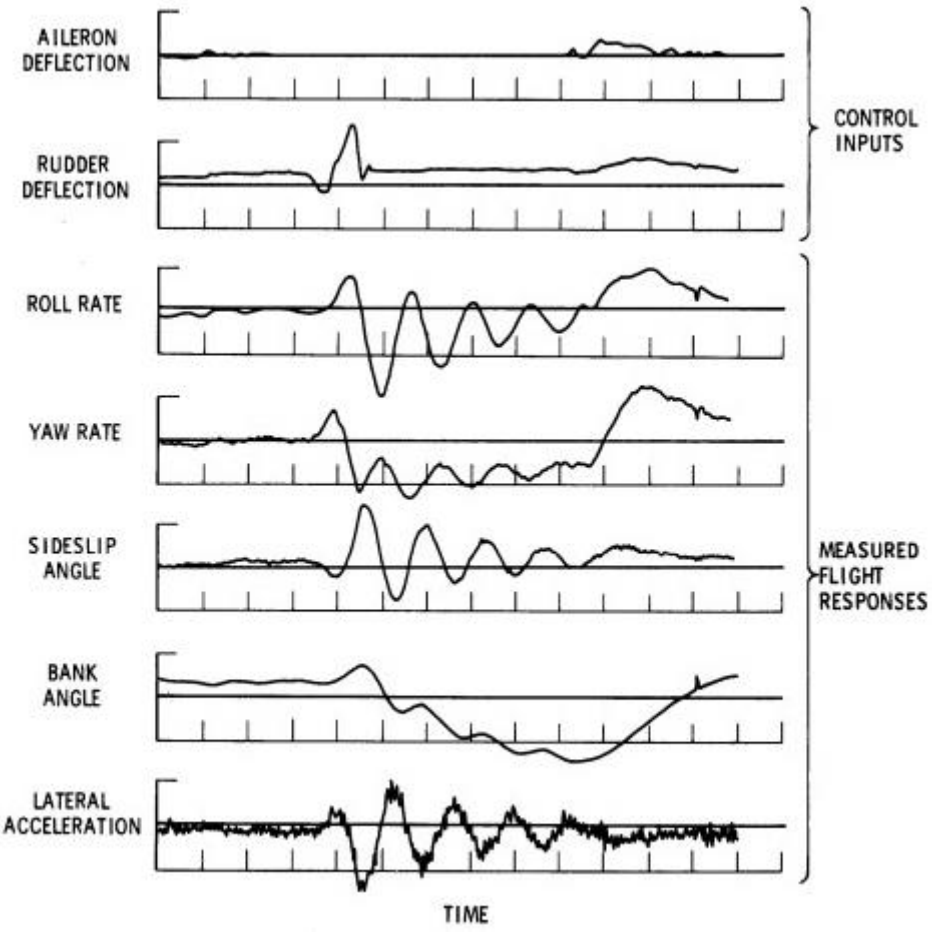


Figure 38: Flight-test data measured for parameter estimation [17]

Once the time histories of all the main parameters have been obtained, a set of six or more linearized equations of motion is used in which the stability and control derivatives values obtained before the flight-tests are initially used. Starting from these values and from the results obtained during the flight-tests, different trial-and-error techniques can be used such as e.g. Ordinary Least Squares, Deterministic Least Squares, Statistical Linearized Filter and Extended Kalman Filter.
Using these techniques, the stability and control derivatives final values are the ones that allow to get as close as possible to the data obtained during flight-tests.

Therefore, flight and computed time histories have almost the exact same trend, as shown in figure:

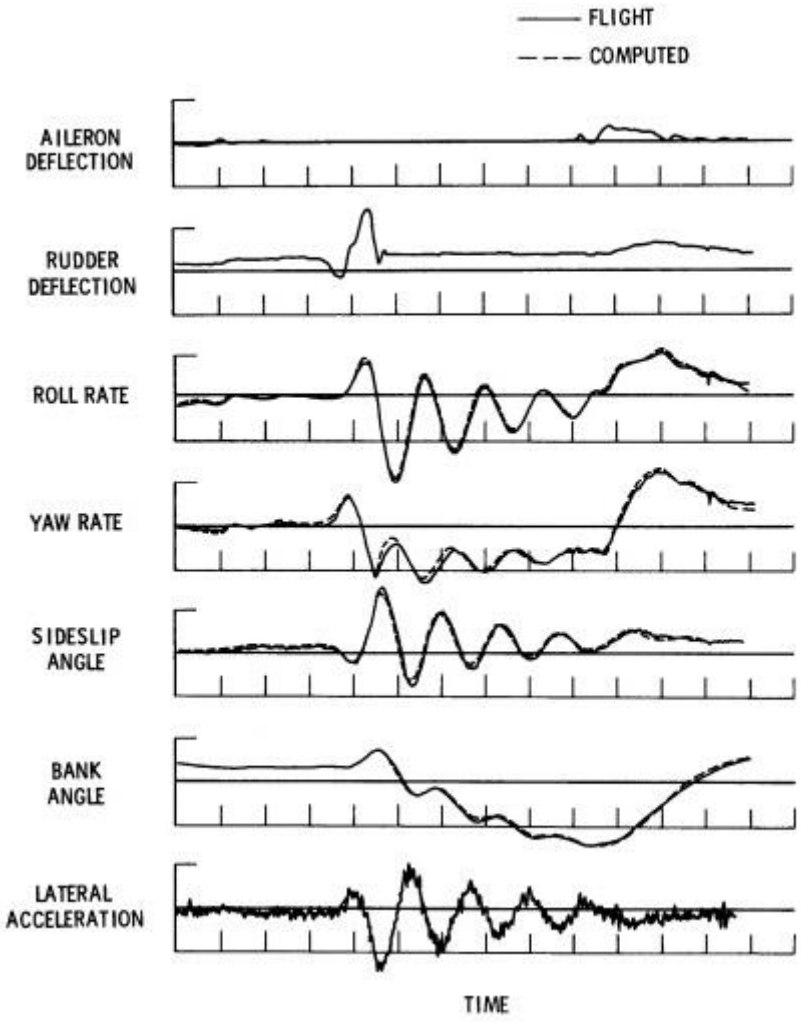


Figure 39: Typical match of computed response using estimated parameter values with the flight-measured response [17]

To get a final estimate, several steps must be performed, consequently these techniques are performed by computers with high computing capacities.

The techniques used for the Parameter Estimation have been improved more and more over the years mainly for two reasons:

- the aircraft performances have improved significantly, consequently their dynamics have changed;

- there is always the goal of having more accurate and efficient techniques, in order to improve the results produced [17].

The Parameter Estimation techniques have five key points:

1. mathematical model;
2. estimation criterion;
3. computational algorithm;
4. total data acquisition system;
5. test input.

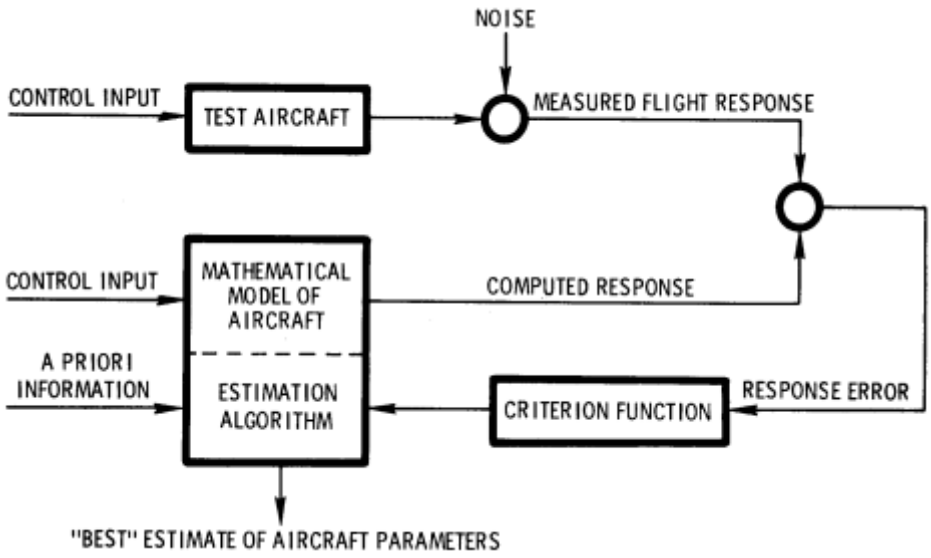


Figure 40: Basic concept of contemporary parameter estimation techniques [17]

A common result to all the Parameter Identification Analysis performed on different types of aircraft and helicopter is that the final values of the stability and control derivatives could be up to 50% different from the initially estimated values.
The final values of these parameters are those that make the simulation more realistic and they are closer to the values obtained during flight-tests. Using these final values, highly effective autopilots and simulators can be implemented.

Some stability derivatives, in fact, can be strongly influenced by phenomena that are difficult to predict during initial phases, such as e.g. the main's rotor's wake that impinges on the tail surfaces. Precisely because of these effects, the Parameter Identification assumes even more importance, because the initial estimate of the stability derivatives can determine not negligible differences compared to the real values. Furthermore, the aircraft/helicopter project can have various evolutions

compared to the first configuration and, in this way, it is possible to obtain values as close as possible to the real ones.

"The correlation coefficient between measured (y) and simulated data (y_{sim}), defined as the normalized cross-covariance function $\rho_{yy_{sim}}$, is given by (Bendat and Piersol [20]*):*

$$\rho_{yy_{sim}} = \frac{\sum_{i=1}^{N}\left[\left(y_i(t) - (1/N)\sum_{i=1}^{N} y_i(t)\right)\left(y_{sim_i}(t) - (1/N)\sum_{i=1}^{N} y_{sim_i}(t)\right)\right]}{\sqrt{\sum_{i=1}^{N}\left[\left(y_i(t) - (1/N)\sum_{i=1}^{N} y_i(t)\right)^2\right]}\sqrt{\sum_{i=1}^{N}\left[\left(y_{sim_i}(t) - (1/N)\sum_{i=1}^{N} y_{sim_i}(t)\right)^2\right]}}$$

can be used to estimate how well the estimated signals can reproduce the measured data. If the correlation coefficient is close to 1, one may conclude that the estimation algorithm can provide a good fit to the experimental data, but on the other hand, if the coefficient is close to 0, the estimation was poor" [18]*.*

N is the simulated outputs number of the proposed model (for example five 3211 maneuvers).

The Parameter Identification is used not only in the **time-domain**, but also in the **frequency-domain**, so frequency-based data are necessary, and they can be obtained even with a conversion of time-based data [21].

6 2nd Application

Carrying out a real Parameter Estimation would require very precise knowledge both in terms of techniques to be used and in terms of data. A good compromise could be to use the same logic to derive an approximation of the fundamental dynamic characteristics, namely:

- **natural frequency f_n**;
- **damping ratio ζ**.

Usually these two parameters are calculated by means of formulas within which there are characteristic coefficients of the aircraft such as aerodynamic coefficients and dimensionless aerodynamic derivatives. These parameters are calculated precisely performing a Parameter Estimation.

The goal of this 2nd application is precisely to obtain quite realistic natural frequency f_n and damping ratio ζ values by analyzing the time histories of the main parameters measured during flight tests.

The main aim is to obtain in a sufficiently short time and in a very simple way an accurate information regarding these two parameters of fundamental importance.

The idea is to calculate them in the following way:

- **natural frequency f_n**: analyzing the period *T* of the various oscillations of the output signal.
- **damping ratio ζ**: analyzing how much the amplitude of the output signal decreases with each oscillation.

6.1 Dynamic Stability

Usually the dynamic stability of a fixed wing aircraft is studied using the theory of small perturbations which allows to decouple the longitudinal plane and the lateral-directional plane.

The dynamic stability of a helicopter, on the other hand, can be studied by decoupling the planes only in the case of hovering or low-speed flight. This difference is because for a helicopter there are couplings between the loads present in the two planes which therefore do not allow to decouple them.

Considering the two coupled planes, the state formulation in compact form is the following:

$$\dot{x} = A \cdot x = \begin{bmatrix} A_{lon} & A_{lat-lon} \\ A_{lon-lat} & A_{lat} \end{bmatrix} \cdot x$$

where $A_{lon-lat}$ and $A_{lat-lon}$ are the two coupling submatrices, in fact, they contain the terms of cross-coupling.
The formulation of the state matrix *A* for full dynamics (coupled planes) is as follows:

$$A = \begin{bmatrix} X_u & X_w & X_q & -gcos\tau_c & X_v & X_p & X_r & 0 & 0 \\ Z_u & Z_w & Z_q + V & -gsin\tau_c & Z_v & Z_p & Z_r & 0 & 0 \\ M_u & M_w & M_q & 0 & M_v & M_p & M_r & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ Y_u & Y_w & Y_q & 0 & Y_v & Y_p & Y_r - V & gcos\tau_c & 0 \\ L'_u & L'_w & L'_q & 0 & L'_v & L'_p & L'_r & 0 & 0 \\ N'_u & N'_w & N'_q & 0 & N'_v & N'_p & N'_r & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

where τ_c is the flight path angle.

The state vector $\boldsymbol{x}$ instead is:

$$x = \{u\ w\ q\ \theta\ v\ p\ r\ \varphi\ \psi\}^T$$

where:

- u, w, q, θ: are the state variables of the longitudinal plane;
- v, p, r, φ, ψ: are the state variables of the lateral-directional plane.

If the helicopter is in hover or in low-speed flight the two planes can be decoupled.

6.1.1 Longitudinal plane

The state-space formulation for the longitudinal plane is:

$$\dot{x} = A \cdot x + B \cdot u$$

$$\begin{Bmatrix} \dot{u} \\ \dot{w} \\ \dot{q} \\ \dot{\theta} \end{Bmatrix} = \begin{bmatrix} X_u & X_w & X_q & -gcos\tau_c \\ Z_u & Z_w & Z_q + V & -gsin\tau_c \\ M_u & M_w & M_q & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{Bmatrix} u \\ w \\ q \\ \theta \end{Bmatrix} + \begin{bmatrix} X_{\theta_0} & X_{B_1} \\ Z_{\theta_0} & Z_{B_1} \\ M_{\theta_0} & M_{B_1} \\ 0 & 0 \end{bmatrix} \cdot \begin{Bmatrix} \theta_0 \\ B_1 \end{Bmatrix}$$

B is the control matrix while u is the command vector where there are the two control variables θ_0 and B_1, respectively the collective command and longitudinal cyclic.

The dynamic stability is studied using the eigenvalue analysis. Once the eigenvalues of the state matrix A have been found, there is a further difference with respect to fixed-wing aircraft: for a fixed-wing aircraft there are two complex conjugated pair of eigenvalues which represent:

- ***Short period mode***: a fast and quite damped dynamic mode;
- ***Long period*** (***phugoid***) ***mode***: a slow and slightly damped dynamic mode.

In fact, the skill of a pilot is to perform a maneuver that triggers only the short period mode and not also the phugoid one.

For a helicopter, instead, there are two real negative eigenvalues and one complex conjugated pair of eigenvalues:

- First real negative eigenvalue represents a damped and aperiodic ***pitch mode***. In hovering the eigenvalue is $\lambda \simeq M_q < 0$ and becomes more and more negative as the speed increases. This dynamic mode is stable thanks the inherent stability of the main rotor, which has a $M_q < 0$ (damping derivative).

- Second real negative eigenvalue represents the ***heave mode***, a damped and aperiodic dynamic mode, that is, the response along the helicopter's vertical axis following a change in the vertical speed w.
  In hovering the eigenvalue is $\lambda \simeq Z_w$, so it depends on the vertical damping derivative Z_w.

- The complex conjugated pair of eigenvalues represent the ***long-period*** (***phugoid***) ***mode***. At very low speeds the respective eigenvalue has a slightly positive real part ($Re(\lambda)>0$) due to the poor efficiency of the tail empennages at low speed, consequently it is a slightly unstable dynamic mode. While starting from slightly higher speeds the real part of the eigenvalue becomes negative ($Re(\lambda)<0$) as the dynamic pressure on the tail empennages increases, consequently their efficiency increases and therefore the phugoid becomes a stable dynamic mode.

6.1.2 Lateral-directional plane

The state-space formulation for the lateral-directional plane is:

$$\dot{x} = A \cdot x + B \cdot u$$

$$\begin{Bmatrix} \dot{v} \\ \dot{p} \\ \dot{r} \\ \dot{\varphi} \\ \dot{\psi} \end{Bmatrix} = \begin{bmatrix} Y_v & Y_p & Y_r - V & g\cos\tau_c & 0 \\ L'_v & L'_p & L'_r & 0 & 0 \\ N'_v & N'_p & N'_r & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{Bmatrix} v \\ p \\ r \\ \varphi \\ \psi \end{Bmatrix} + \begin{bmatrix} Y_{A_1} & Y_{\theta_{tr}} \\ L'_{A_1} & L'_{\theta_{tr}} \\ N'_{A_1} & N'_{\theta_{tr}} \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \cdot \begin{Bmatrix} A_1 \\ \theta_{tr} \end{Bmatrix}$$

where A_1 is the lateral cyclic while θ_{tr} is the pedal command.

The aerodynamic derivatives are always obtained with the decomposition in principal axes of inertia: in this way all the rotations are around the principal axes of inertia.

The principal axes of inertia do not coincide with the body axes because of I_{XZ}, but while for a fixed-wing aircraft I_{XZ} is at least two orders of magnitude smaller than I_{XX} and I_{ZZ}, for a helicopter it is high because of:

- Tail rotor;
- Tail empennages;
- Vertical and not uniform mass distribution.

Therefore, I_{XZ} is only one order of magnitude smaller than I_{XX} and I_{ZZ}, consequently when the pilot gives a lateral cyclic command, in addition to the roll response, there is also a yaw one because the moment of inertia I_{XZ} couples the roll response to the yaw response.

The apexes " ' " are therefore due to keep in consideration the I_{XZ} moment of inertia and its coupling effect:

$$L'_i = \frac{L_i + \frac{I_{XZ}}{I_{XX}} N_i}{1 - \frac{I_{XZ}^2}{I_{XX} I_{ZZ}}} \simeq L_i + \frac{I_{XZ}}{I_{XX}} N_i$$

$$N'_i = \frac{N_i + \frac{I_{XZ}}{I_{ZZ}} L_i}{1 - \frac{I_{XZ}^2}{I_{XX} I_{ZZ}}} \simeq N_i + \frac{I_{XZ}}{I_{ZZ}} L_i$$

There are five eigenvalues λ extracted from the state matrix A, but one of them is null ($\lambda = 0$) and it represents the heading mode. The other four eigenvalues are:

- A complex conjugated pair of eigenvalues for the ***dutch roll*** that have a real positive part ($Re(\lambda) > 0$) at low speeds due to the poor efficiency of the tail empennages and tail rotor at such speeds, consequently FCS (Flight Control System) is used to stabilize this dynamic mode at these speeds. Increasing the flight speed, their efficiency raises, so the *dutch roll* became a stable dynamic mode ($Re(\lambda) < 0$).

- A real negative eigenvalue for the ***roll mode***, that is an aperiodic stable dynamic mode thanks to the main rotor. In fact, this is a symmetric case of the pitch mode. In hovering the eigenvalue is $\lambda \simeq L_P$.

- A real negative eigenvalue for the ***spiral mode***. For a helicopter the *spiral mode* is always a stable dynamic mode thanks to the $N_r < 0$ yaw damping derivative that depends on the tail rotor characteristics. In fact, in hover conditions, the eigenvalue is $\lambda \simeq N_r < 0$. For a fixed-wing aircraft, instead, the *spiral mode* eigenvalue is $Re(\lambda) \simeq 0$, so the real part of the eigenvalue switch from positive to negative values and vice versa because of the dihedral effect, so it could be unstable [22].

6.2 2nd Application Implementation

For the 2nd application a new C++ code is necessary because only the Spatial will be used, not the camera. In fact, using the Spatial, the fundamental variables described previously can be measured, as the Euler's angles, angular velocity and the acceleration along the body axes.

Plotting the time history of each variable it will be possible to verify if, following a certain pilot input command maneuver, the helicopter response is such that the induced oscillations tend to decrease quite quickly.
The application aim is to get some approximative values but still quite realistic of the natural frequency f_n and the damping ratio ζ.

The same approach used for the 1st application was used to realize the graphs. Even in this case two windows were created: one for the longitudinal plane variables and the other one for the lateral-directional plane variables. This choice, as well as the chosen variable, can be obviously modified according to the user's needs.

The user can switch from a video screen to another through the new buttons inserted near the ones previously described:

- ***NEXT PAGE button***: this button is present in the longitudinal plane window and it allows to switch to the lateral-directional plane window.

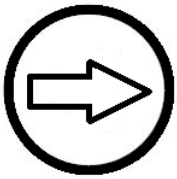


Figure 41: NEXT PAGE button icon

- ***PREVIOUS PAGE button***: this button is present in the lateral-directional plane window and it allows to switch to the longitudinal plane window.

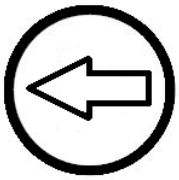


Figure 42: PREVIOUS PAGE button icon

Even in this application the graphs are created from when the user starts the software, but the several data values are saved on a text file only from when the user presses on the *PLAY* button until he presses the *PAUSE* or *EXIT* button.

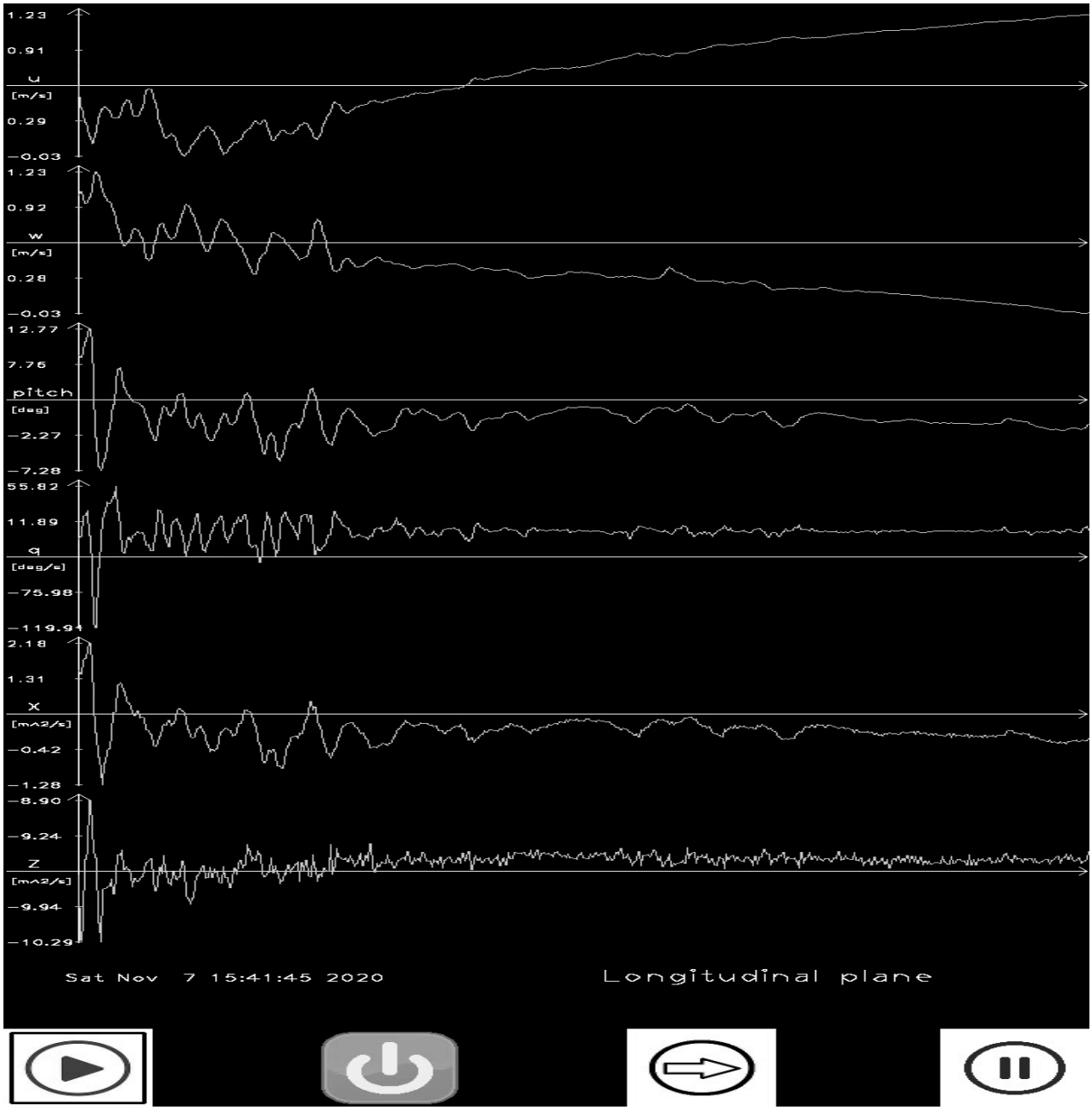


Figure 43: Longitudinal plane window

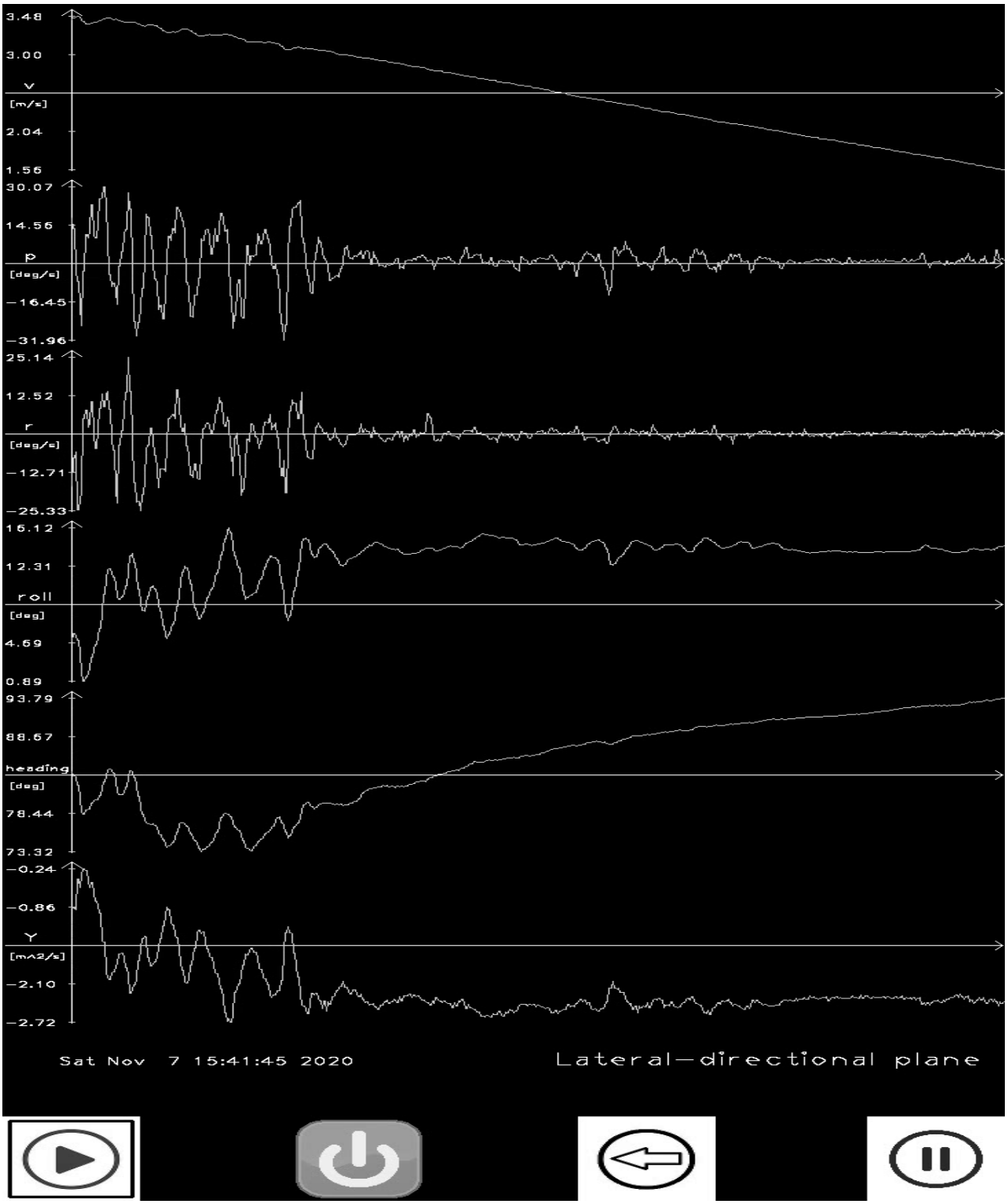


Figure 44: Lateral-Directional plane window

The user has full freedom in choosing which of the two screens to display at that moment using the two buttons *NEXT PAGE* and *PREVIOUS PAGE*, without this choice being able to modify or generate errors in the creation of the graphs in real time and in saving the various data on the text file.

Moreover, in order to save two different videos (one for each display), another thread *save_video* was created: so, when the user click to the *PLAY* button, two videos are created and saved.
Consequently, another function called *save_video_lat_dir* was created in order to save the video of the Lateral-Directional plane window.

```
// Initialize and set thread joinable
pthread_attr_init(&attr);
pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);


// create thread for getting IMU
pthread_create(&thread_id_geo,&attr,getGeoInfo,(void *)&GEO);

// create thread to save longitudinal plane video
pthread_create(&thread_id_save_video, &attr, save_video, (void
*)&VIDEO);

// create thread to save lateral-directional plane video
pthread_create(&thread_id_save_video_lat_dir, &attr,
save_video_lat_dir, (void *)&VIDEO);


SensorFusion( &GEO, &VIDEO );


// close thread for getting IMU
pthread_join(thread_id_geo, &status);

// close thread to save longitudinal plane video
pthread_join(thread_id_save_video, &status);

// close thread to save lateral-directional plane video
pthread_join(thread_id_save_video_lat_dir, &status);


pthread_attr_destroy(&attr);
```

In this way, for example, if the user has seen the Longitudinal plane window during the saving phase, he can also be able to view the Lateral-directional plane window during the post-processing phase.
The window chosen by the user during the saving phase does not affect in any way the saving of the two videos and does not generate any type of error.
Consequently, the user has full freedom in choosing a specific window rather than another in the registration phase.

6.3 Spatial Manager

From the Spatial web site [23] it is possible to download the *Spatial Manager*, available for Mac, Windows and Linux. This is a graphic interface of the Spatial.

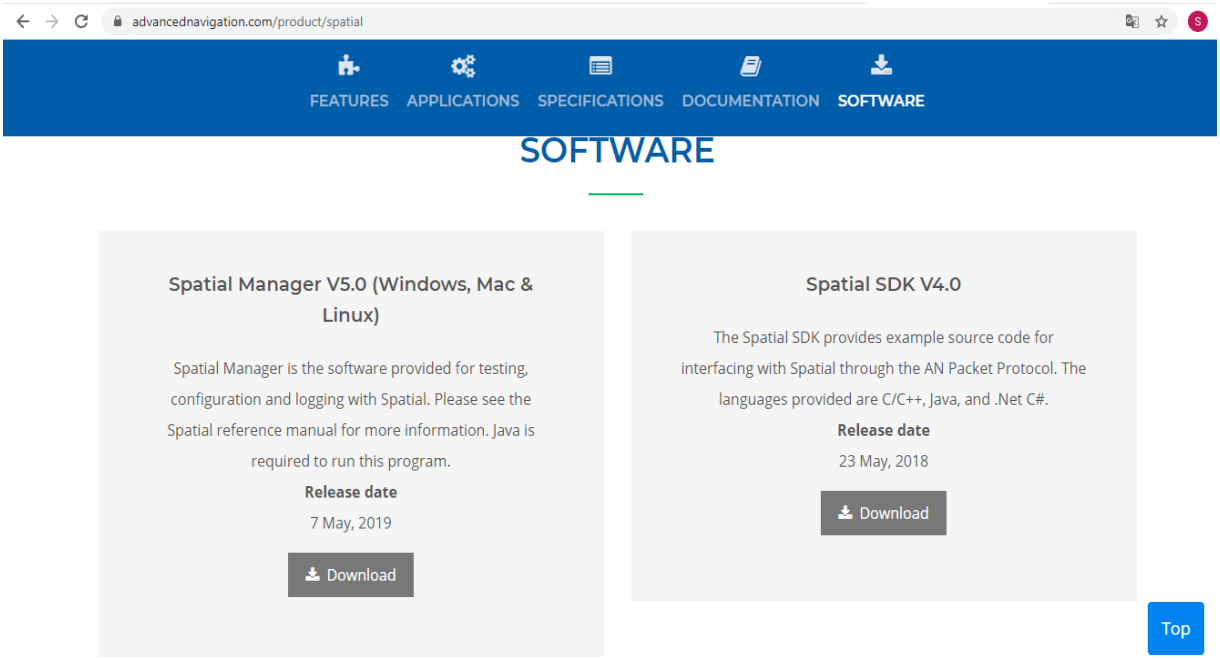


Figure 45: Spatial web site [23]

An update version of Java is necessary to use the *Spatial Manager*.

After downloading Java, it is recommended to use the following command in the Linux Terminal Command Line to avoid having some permissions problems:

sudo adduser username dialout

To execute the *.jar* file and finally use the *Spatial Manager* it is necessary to use the *cd* command to go in the *.jar* file directory and the last command to use is the following:

sudo java -jar SpatialManager-5.0.jar

After using this command, the *Spatial Manager* software will start.

The Spatial SDK codes allow to use several packets, each of one allows to get different data types as angles, velocity in body axes / NED axes, angular velocity in body axes, accelerations in body axes, angular accelerations, latitude, longitude, temperature and pressure on the Spatial, g-force, etc.

To activate and/or deactivate the various packets, each distinguishable by its own ID, and to change the rate used to get the data values, it is necessary to use the *Spatial Manager*. In fact, these changes cannot be carried out by the C/C++ codes.

After starting *Spatial Manager* and connecting the Spatial to the pc, the following screen appears:

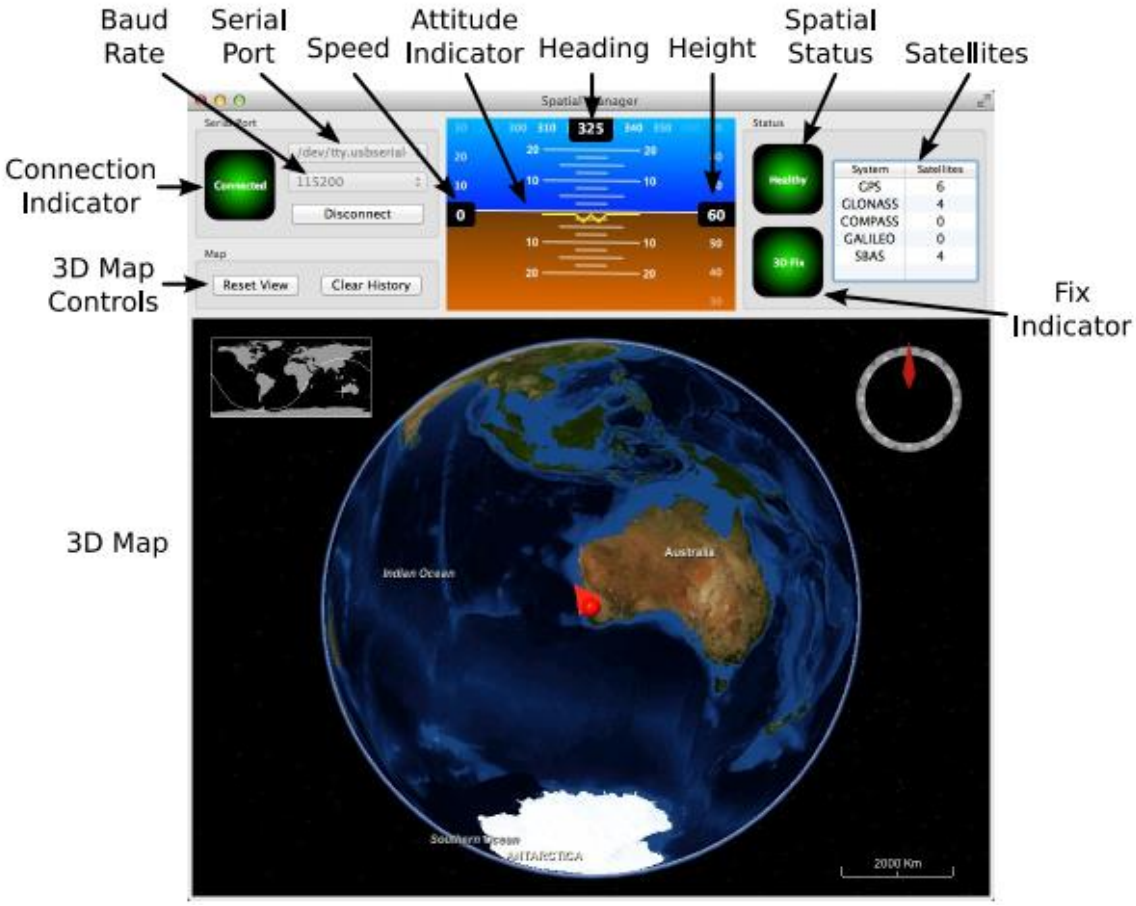


Figure 46: Spatial Manager graphic interface [24]

A good pc graphic card is necessary to view the 3D map, otherwise this screen section will appear completely black.

The top center screen is very similar to the aircraft Primary Flight Display (PFD), in fact there are the heading, height, speed and attitude information. There are also the artificial horizon and the same colors (brown and blue) which are used on the PFD.

To activate and/or deactivate packets the user must go in *Configuration -> Packet Rates* and must add *ID* and *Period* for each packet that he wants to use.
In the *Spatial Reference Manual* [24] it is possible to find all the packets *IDs*.

The System State Packet (ID 20) and the Raw Sensors Packet (ID 28) are the default packet enabled at 50Hz and these typically provide almost all the data that a user will require. However, it is possible to change the output rate changing the *Period*.

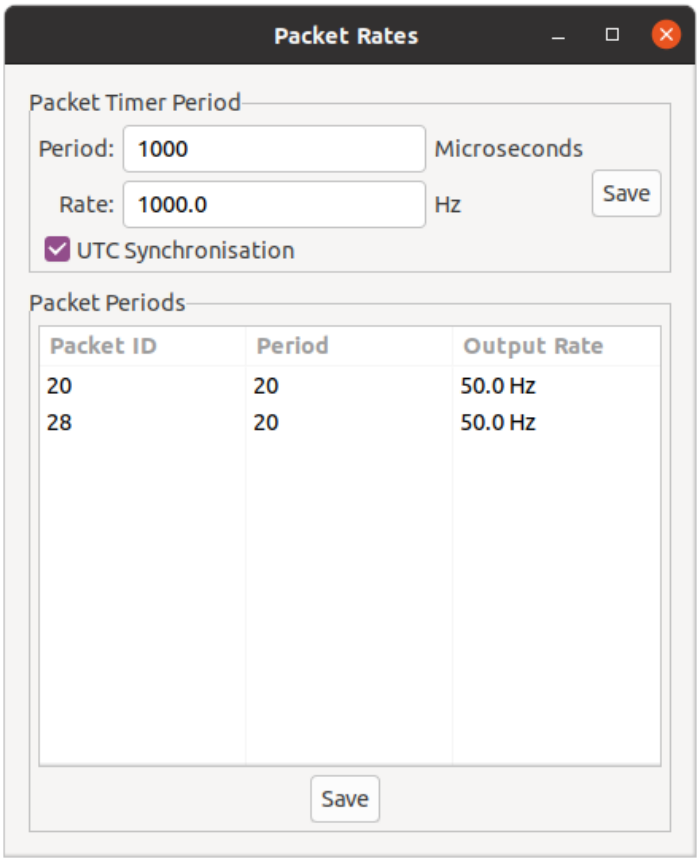


Figure 47: Packet Rates - Spatial Manager

Before increasing the packet rates, it is essential to ensure that the baud rate is adequate to handle the data throughput. By default, the Spatial is configured with a baud rate of 115200.
The baud rate can be calculated using the packet rate and size.
The packet size is equal to the packet length add five to account for the packet overhead.
By default, the packet rate is 50 Hz, so, considering that the System State Packet length is 100, it is necessary to do the following calculations:

Data throughput = [100 (packet length) + 5 (fixed packet overhead)] x 50 (rate)
Data throughput = 5250 bytes per second

Minimum baud rate = Data throughput x 11 = 57750 Baud
Closest standard baud rate = 115200 Baud

Using these formulas, it is possible to calculate the necessary baud rate for each packet rate that the user wants to use.
When the user wants to use multiple packets at the same rate, he must order them from the lowest ID to the highest ID.

Once the necessary packets are activated, some changes on *getGeoInfo* function are necessary in order to get the desired data:

```
an_decoder_t an_decoder;
an_packet_t *an_packet;

system_state_packet_t system_state_packet;
raw_sensors_packet_t raw_sensors_packet;
body_velocity_packet_t body_velocity_packet;

...

else if (an_packet->id == packet_id_raw_sensors) /* raw sensors packet
*/
{
/* copy all the binary data into the typedef struct for the packet */
/* this allows easy access to all the different values             */
    if(decode_raw_sensors_packet(&raw_sensors_packet, an_packet) == 0)
    {
            mtx_IMU.lock();
            GEO->IMU_SRC.acc_x = raw_sensors_packet.accelerometers[0];
            GEO->IMU_SRC.acc_y = raw_sensors_packet.accelerometers[1];
            GEO->IMU_SRC.acc_z = raw_sensors_packet.accelerometers[2];

            GEO->flag_acc = 1;
            mtx_IMU.unlock();
    }
}
```

This one is an example of how call back a packet in the C++ code and how to use the SDK. For each packet used it is necessary to add these simple command lines.

6.4 Packet Summary

The Spatial packets available are the following:

Packet ID	Length	R/W	Name
System Packets			
0	4	R	Acknowledge Packet
1	-	W	Request Packet
2	1	R/W	Boot Mode Packet
3	24	R	Device Information Packet
4	4	W	Restore Factory Settings Packet
5	4	W	Reset Packet
10	-	R/W	Serial Port Pass-through Packet
State Packets			
20	100	R	System State Packet
21	8	R	Unix Time Packet
22	14	R	Formatted Time Packet
23	4	R	Status Packet
24	12	R	Position Standard Deviation Packet
25	12	R	Velocity Standard Deviation Packet
26	12	R	Euler Orientation Standard Deviation Packet
27	16	R	Quaternion Orientation Standard Deviation Packet
28	48	R	Raw Sensors Packet
29	74	R	Raw GNSS Packet
30	13	R	Satellites Packet
31	-	R	Detailed Satellites Packet
32	24	R	Geodetic Position Packet
33	24	R	ECEF Position Packet
34	26	R	UTM Position Packet
35	12	R	NED Velocity Packet
36	12	R	Body Velocity Packet
37	12	R	Acceleration Packet
38	16	R	Body Acceleration Packet
39	12	R	Euler Orientation Packet
40	16	R	Quaternion Orientation Packet
41	36	R	DCM Orientation Packet
42	12	R	Angular Velocity Packet
43	12	R	Angular Acceleration Packet
44	60	R/W	External Position & Velocity Packet
45	36	R/W	External Position Packet
46	24	R/W	External Velocity Packet
47	16 or 24	R/W	External Body Velocity Packet

48	8	R/W	External Heading Packet
49	8	R	Running Time Packet
50	12	R	Local Magnetic Field Packet
51	20	R	Odometer State Packet
52	8	R/W	External Time Packet
53	8	R/W	External Depth Packet
54	4	R	Geoid Height Packet
55	-	W	RTCM Corrections Packet
56	-	-	External Pitot Pressure Packet
57	12	R/W	Wind Packet
58	16	R	Heave Packet
59	-	-	Post Processing Packet
60	-	R	Raw Satellite Data Packet
67	13	R/W	External Odometer Packet
68	25	R/W	External Air Data Packet
72	8	R/W	Gimbal State Packet
73	24	R	Automotive Packet
Configuration Packets			
180	4	R/W	Packet Timer Period Packet
181	-	R/W	Packets Period Packet
182	17	R/W	Baud Rates Packet
184	4	R/W	Sensor Ranges Packet
185	73	R/W	Installation Alignment Packet
186	17	R/W	Filter Options Packet
187	-	-	Advanced Filter Parameters Packet
188	13	R/W	GPIO Confguration Packet
189	49	R/W	Magnetic Calibration Values Packet
190	1	W	Magnetic Calibration Confguration Packet
191	3	R	Magnetic Calibration Status Packet
192	8	R/W	Odometer Confguration Packet
193	5	W	Set Zero Orientation Alignment Packet
194	49	R/W	Reference Point Offsets Packet
195	33	R/W	GPIO Output Confguration Packet
198	64	R/W	User Data Packet
199	65	R/W	GPIO Input Confguration Packet

Figure 48: Spatial Packets [24]

For the 2nd application the Body Velocity Packet (ID 36) was added because it is necessary to get the velocity in body axes (*X, Y, Z*) *u, v, w*.

The default packets *System State Packet* (ID 20) and *Raw Sensors Packet* (ID 28) allow to get the following data values [24]:

System State Packet				
Packet ID			20	
Packet Length			100	
Field #	**Bytes Offset**	**Data Type**	**Size**	**Description**
1	0	u16	2	System status, see section 13.9.1.1
2	2	u16	2	Filter status, see section 13.9.1.2
3	4	u32	4	Unix time (seconds), see section 13.9.1.4
4	8	u32	4	Microseconds, see section 13.9.1.5
5	12	fp64	8	Latitude (rad)
6	20	fp64	8	Longitude (rad)
7	28	fp64	8	Height (m)
8	36	fp32	4	Velocity north (m/s)
9	40	fp32	4	Velocity east (m/s)
10	44	fp32	4	Velocity down (m/s)
11	48	fp32	4	Body acceleration X (m/s/s)
12	52	fp32	4	Body acceleration Y (m/s/s)
13	56	fp32	4	Body acceleration Z (m/s/s)
14	60	fp32	4	G force (g)
15	64	fp32	4	Roll (radians)
16	68	fp32	4	Pitch (radians)
17	72	fp32	4	Heading (radians)
18	76	fp32	4	Angular velocity X (rad/s)
19	80	fp32	4	Angular velocity Y (rad/s)
20	84	fp32	4	Angular velocity Z (rad/s)
21	88	fp32	4	Latitude standard deviation (m)
22	92	fp32	4	Longitude standard deviation (m)
23	96	fp32	4	Height standard deviation (m)

Figure 49: System State Packet [24]

Raw Sensors Packet				
Packet ID		28		
Packet Length		48		
Field #	**Bytes Offset**	**Data Type**	**Size**	**Description**
1	0	fp32	4	*Accelerometer X (m/s/s)*
2	4	fp32	4	*Accelerometer Y (m/s/s)*
3	8	fp32	4	*Accelerometer Z (m/s/s)*
4	12	fp32	4	*Gyroscope X (rad/s)*

5	16	fp32	4	*Gyroscope Y (rad/s)*
6	20	fp32	4	*Gyroscope Z (rad/s)*
7	24	fp32	4	*Magnetometer X (mG)*
8	28	fp32	4	*Magnetometer Y (mG)*
9	32	fp32	4	*Magnetometer Z (mG)*
10	36	fp32	4	*IMU Temperature (deg C)*
11	40	fp32	4	*Pressure (Pascals)*
12	44	fp32	4	*Pressure Temperature (deg C)*

Figure 50: Raw Sensors Packet [24]

6.5 Damping Ratio and Natural Frequency

Previously, the various dynamic ways in which a helicopter responds to a certain command given by the pilot have been described. The periodic dynamic modes are characterized by a natural frequency f_n and a damping ratio ζ.
If $0 < \zeta < 1$ the dynamic mode is underdamped, while if $\zeta > 1$ it is overdamped.

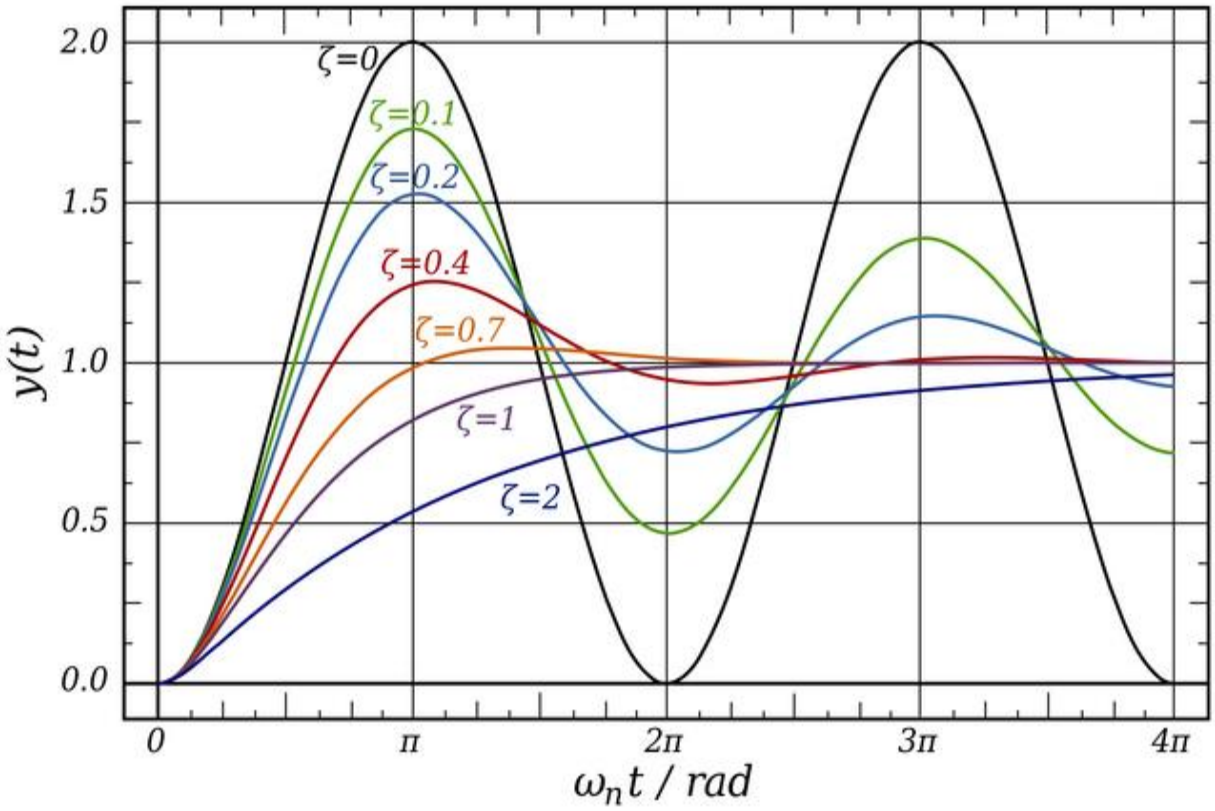


Figure 51: Output response according to damping ratio value [25]

If $\zeta = 1$ the output response is critically damped, so it reaches steady state value as quickly as possible without being weakened, while if $\zeta > 1$ the response does not oscillate around the steady state value but takes longer to reach the steady state than the critically damped case.

The aim of this 2nd application is to find an approximation of the natural frequency f_n and damping ratio ζ in a simple way, analyzing the output responses obtained after a command input given by the pilot.

Considering the following figure representing a capture of the longitudinal plane window, it is possible to notice that the output response is underdamped ($\zeta < 1$) because of its decrease in amplitude over time and of its oscillation around a stationary value.

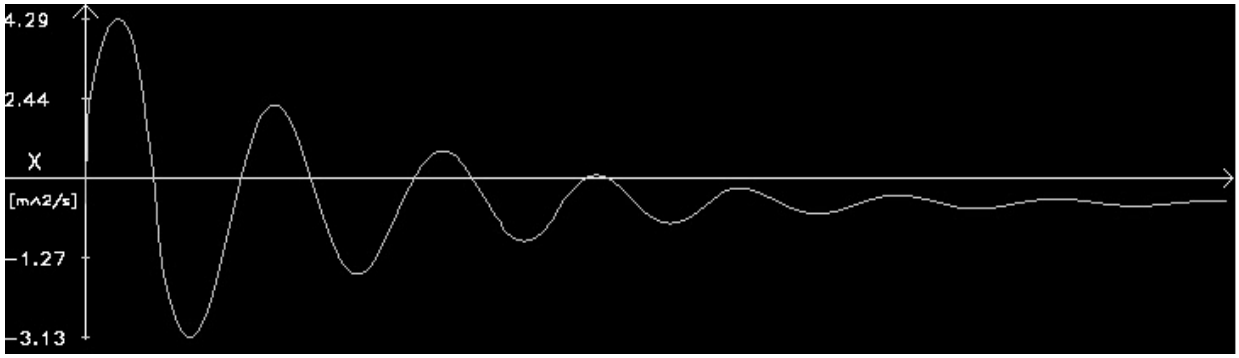


Figure 52: X acceleration output signal

To facilitate the work of estimating the natural frequency f_n and the damping ratio ζ, it is possible to use in a MATLAB code the data saved in output in the *.xlsx* file.

The same curve represented on MATLAB is the following:

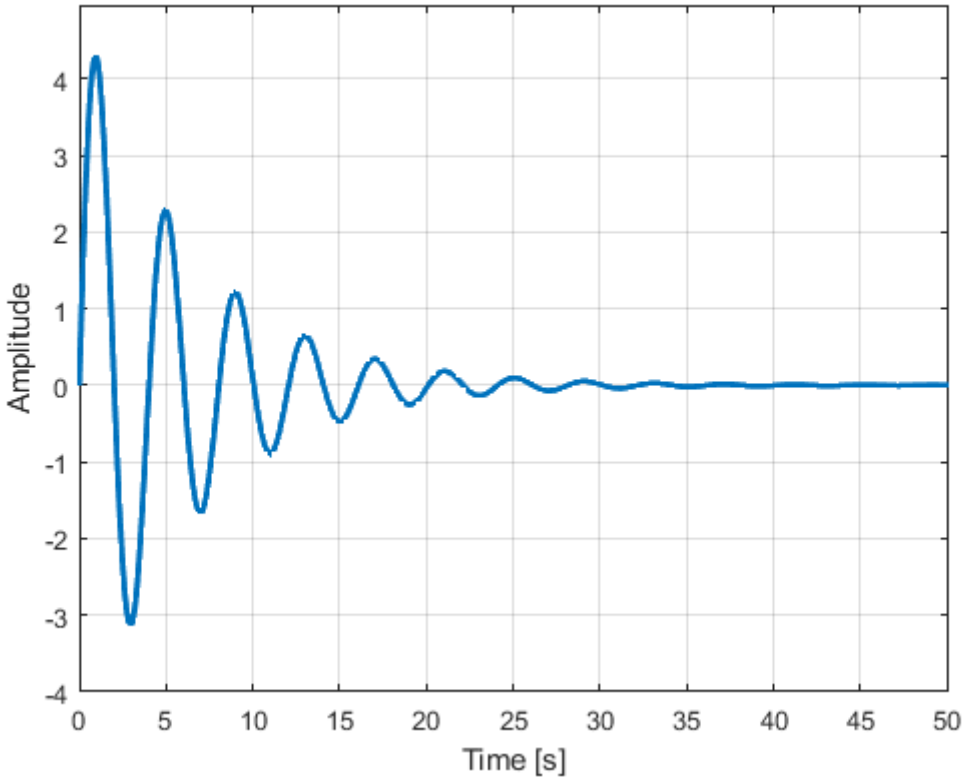


Figure 53: X acceleration curve on MATLAB

By representing this curve on MATLAB, it is possible to see its complete trend over time.

The natural frequency f_n can be calculated by inverse the time elapsed between one peak and the next one.

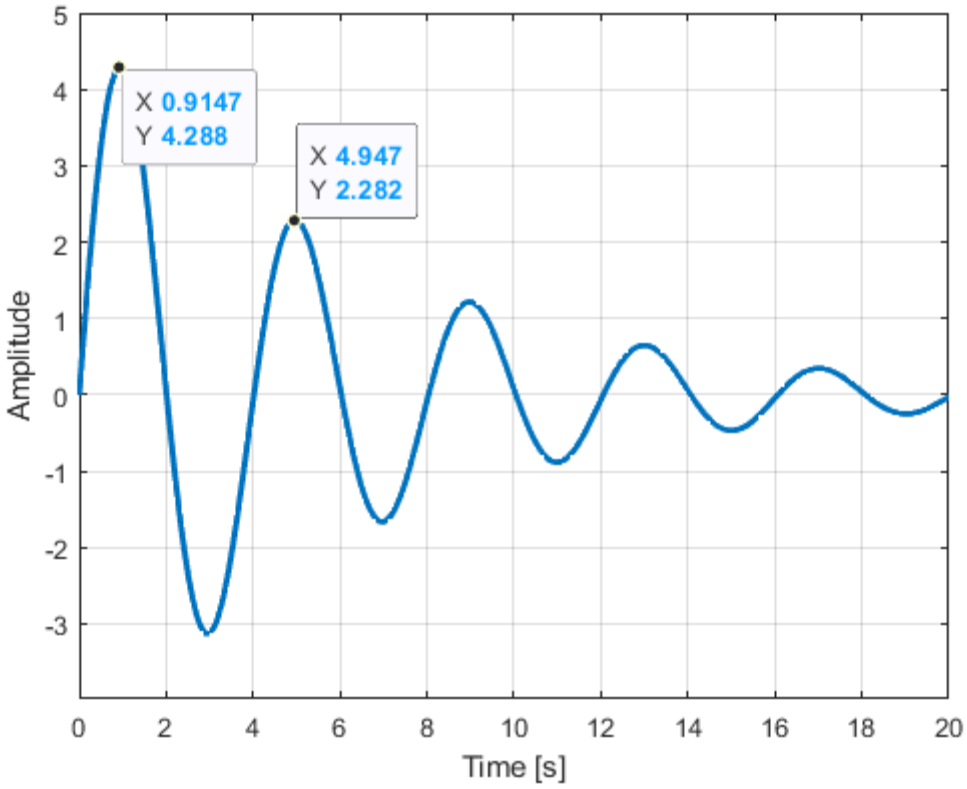


Figure 54: Time instant of the peaks

The first peak occurs at the time instant $t = 0.9147\,s$, while the next one at $t = 4.947\,s$, therefore the time elapsed between one peak to the next one is equal to $T_d = 4.0323\,s$ and the damped natural frequency is $f_d = 1/T_d = 0.248\,Hz$.

The undamped natural frequency is equal to:

$$f_n = f_d \cdot \sqrt{1 - \zeta^2}$$

Therefore, in order to obtain the natural frequency f_n, it is necessary to calculate the damping ratio ζ.

Obviously, this curve is just an example, i.e. the values present must not be considered as real values obtained from a flight test on board an aircraft.

Moreover, it is possible to calculate the damping ratio ζ in different ways: the simplest one is to use the **Logarithmic Decrement method** which exploits the knowledge of the amplitudes of two peaks to finally obtain a damping ratio value. This method can be used only for an underdamped system ($\zeta < 1$) and it becomes less and less precise for $0.5 < \zeta < 1$ [26].

Analyzing the time response of an underdamped vibration it is possible to calculate the logarithmic decrement δ that represents the reduction rate of the amplitude of a free dumped oscillation.

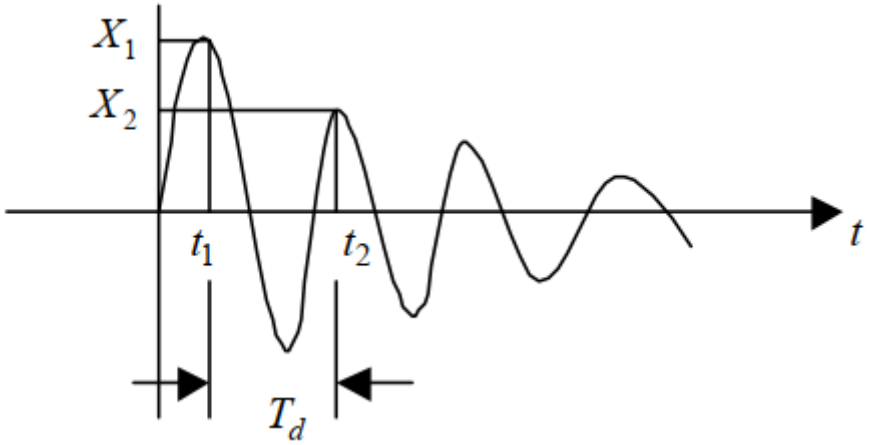


Figure 55: Time response of an underdamped oscillation

δ is equal to the natural logarithm of the ratio of any two successive amplitudes.
Considering the time response of an underdamped vibration $x(t)$ it is possible to obtain, after some steps, a formula for the logarithmic decrement δ:

$$x(t) = Ae^{-\zeta\omega_n t}cos(\omega_d t - \phi)$$

where $\omega_n = 2\pi \cdot f_n$ is the natural pulsation, ζ is the damping ratio and $\omega_d = \omega_n \cdot \sqrt{1-\zeta^2}$ is the natural damped pulsation.

Considering two instants time t_1 and t_2, the relative amplitudes are:

$$X_1 = Ae^{-\zeta\omega_n t_1} \qquad X_2 = Ae^{-\zeta\omega_n t_2} = Ae^{-\zeta\omega_n (t_1+T_d)}$$

The ratio between these two amplitudes is:

$$\frac{X_1}{X_2} = \frac{Ae^{-\zeta\omega_n t_1}}{Ae^{-\zeta\omega_n (t_1+T_d)}} = e^{\zeta\omega_n T_d}$$

Therefore, the logarithmic decrement δ is by definition:

$$\delta = \ln\frac{X_1}{X_2} = \zeta\omega_n T_d = \zeta\omega_n \frac{2\cdot\pi}{\omega_n\sqrt{1-\zeta^2}} = \frac{2\cdot\pi\cdot\zeta}{\sqrt{1-\zeta^2}}$$

Finally, it is possible to calculate the damping ratio ζ reversing the previous formula:

- If $\zeta \ll 1$ ($\zeta < 0.1$):

$$\zeta = \frac{\delta}{2 \cdot \pi}$$

- If $\zeta > 0.1$:

$$\zeta = \frac{\delta}{\sqrt{4 \cdot \pi^2 + \delta^2}}$$

The same procedure can be performed considering any pair of amplitude values, e.g. X_1 and X_{N+1}.
In this case the second amplitude used is:

$$X_{N+1} = Ae^{-\zeta\omega_n(t_1+N\cdot T_d)}$$

The amplitude ratio is:

$$\frac{X_1}{X_{N+1}} = \frac{Ae^{-\zeta\omega_n t_1}}{Ae^{-\zeta\omega_n(t_1+N\cdot T_d)}} = e^{\zeta\omega_n N T_d}$$

and the natural logarithm of this ratio is:

$$\ln\frac{X_1}{X_{N+1}} = N \cdot (\zeta\omega_n T_d) = N \cdot \delta$$

Therefore, the logarithmic decrement δ is equal to:

$$\delta = \frac{1}{N}\ln\frac{X_1}{X_{N+1}}$$

This is a general formula to calculate δ [27].

Using the Logarithmic Decrement Method in the previous case it is possible to obtain the damping ratio ζ:

$$\delta = \ln\frac{4.288}{2.282} = 0.63$$

$$\zeta = \frac{\delta}{\sqrt{4 \cdot \pi^2 + \delta^2}} = \frac{0.63}{\sqrt{4 \cdot \pi^2 + 0.63^2}} = 0.1 \quad \Rightarrow \quad f_n = \frac{f_d}{\sqrt{1-\zeta^2}} = 0.25\ Hz$$

There are also other methods to calculate the damping ratio ζ, e.g. it is possible to obtain it in an iterative way finding a **decreasing exponential curve** that envelops the response.
This exponential curve is of the following type:

$$\mathrm{f}(t) = \mathrm{A} \cdot e^{-\zeta\omega_n t} = A \cdot e^{-\zeta \cdot \frac{2\pi f_d}{\sqrt{1-\zeta^2}} t}$$

and this curve passes through each peak of the response [28].

However, in this function there are two unknowns:

- Damping ratio ζ;
- Maximum amplitude A.

The problem of the 2nd unknown can be easily fixed considering the value of the 1st peak and the time when it occurs: from *figure 53* can be noticed that $y = 4.288$ and $\mathrm{t}_{\mathrm{y_{max}}} = 0.91\,s$.
Therefore, the exponential curve can be rewritten in this way:

$$f\left(t + t_{y_{max}}\right) = 4.288 \cdot e^{-\zeta \frac{2\pi f_d}{\sqrt{1-\zeta^2}} \cdot (t + t_{y_{max}})}$$

Finally, using $f_d = 0.248\,Hz$ (calculated previously) and $\zeta = 0.1$, it is possible to obtain the correct decreasing exponential curve:

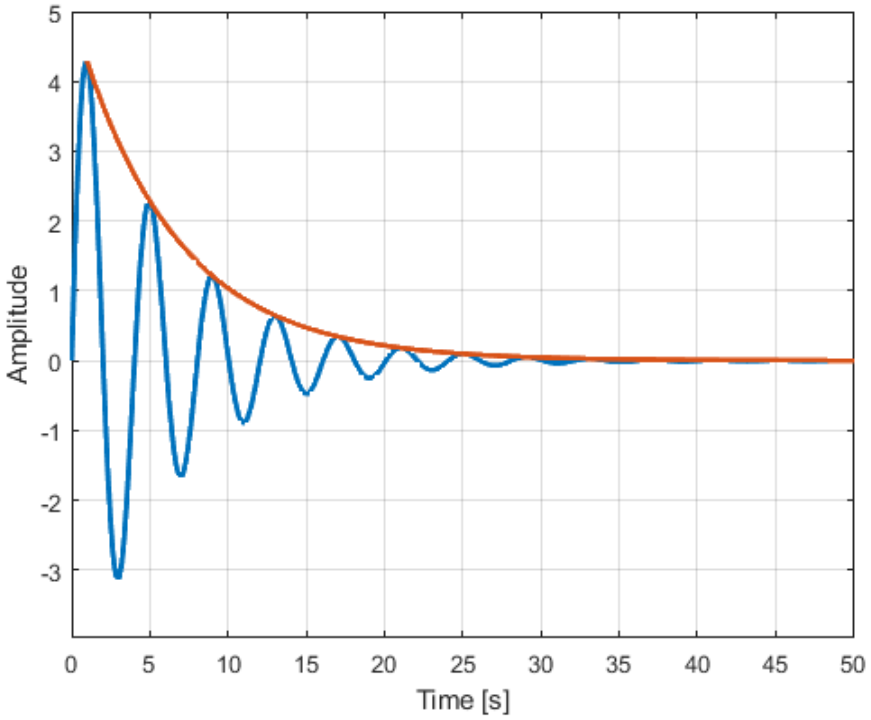


Figure 56: Decreasing Exponential curve

Finally, the undamped natural frequency can be calculated:

$$f_n = \frac{f_d}{\sqrt{1-\zeta^2}} = 0.25\,Hz$$

Another method to calculate both ω_n and ζ is to use the **Least Squares Method** to find the function that fits the set of data saved in the *.xlsx* file and that has the following kind of equation:

$$\mathrm{f}(t) = F_\infty + \mathrm{A}\cdot e^{-\zeta\cdot 2\pi\cdot f_n}\cdot c\cos\left(2\pi\cdot f_n\cdot\sqrt{1-\zeta^2}\cdot \mathrm{t}+\phi\right)$$

where F_∞ is the steady state value of the function.

Once the equation of the function in this form is obtained, it is possible to derive immediately the natural frequency f_n and the damping ratio ζ.

The following MATLAB script [29] can be used to obtain them, considering the first 1000 values of the dataset:

```
time = tbl.VarName1;
signal = tbl.VarName6;

y = signal(1:1000, 1);
x = time(1:1000, 1);

yu = max(y);
yl = min(y);
yr = (yu-yl);                               % Range of 'y'
yz = y-yu+(yr/2);
zx = x(yz .* circshift(yz,[1 0]) <= 0);     % Find zero-
crossings
per = 2*mean(diff(zx));                     % Estimate period
ym = mean(y);                               % Estimate offset

% Function to fit
fit = @(b,x)  b(1).*(cos(2*pi*x./b(2) + 2*pi/b(3))) .*
exp(b(4).*x) + b(5);

% Sum-squared-error cost function
fcn = @(b) sum((fit(b,x) - y).^2);
```

```
% Minimise Least-Squares
s = fminsearch(fcn, [yr;  per;  -1;  -1;  ym], options)

xp = linspace(min(x),max(x));

figure(1)
plot(x,y,'b', 'LineWidth',1)
hold on
plot(xp,fit(s,xp), '--r', 'LineWidth',1.25)
hold off
grid
text(5, 3, sprintf('%.2f\\cdot cos(2\\cdot \\pi\\cdot
%.4f\\cdot x%+.2f\\cdot 2\\cdot\\pi)\\cdot e^{%.4f\\cdot x}
+ %.2f', s(1),1/s(2),1/s(3),s(4),s(5)))

legend('Data', 'Regression')
```

The final figure obtained is the following, in which the equation of the curve obtained through the **curve fitting** is also transcribed:

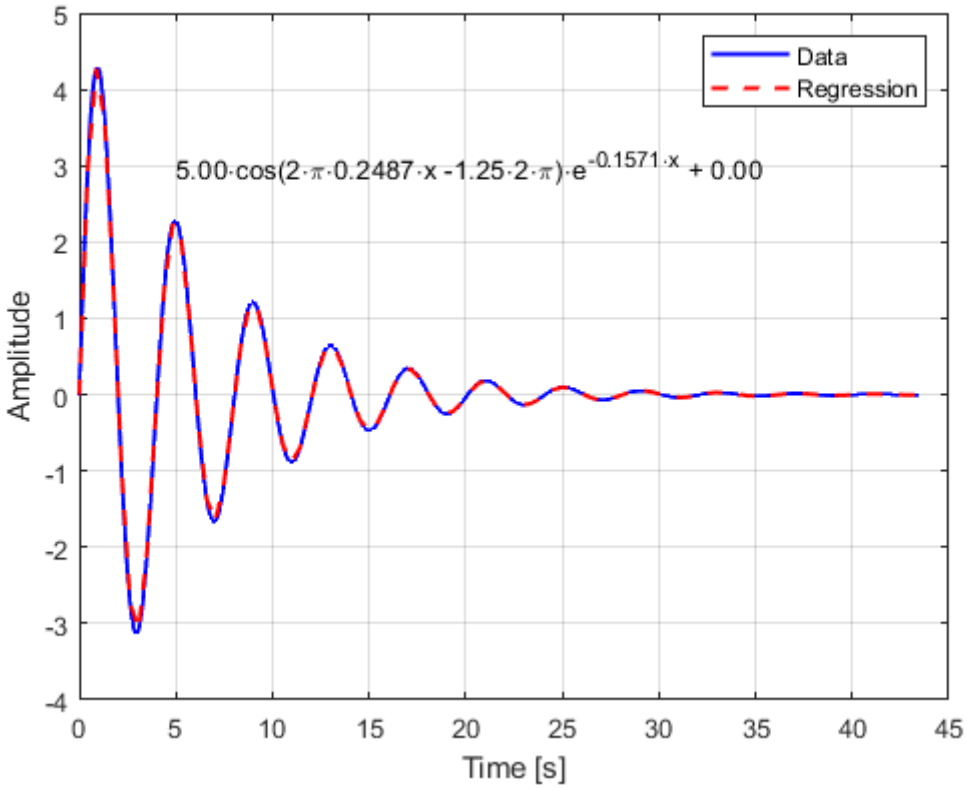


Figure 57: Curve fitting

Therefore:

$$\begin{cases} f_n \cdot \sqrt{1-\zeta^2} = 0.2487 \\ \zeta \cdot 2\pi \cdot f_n = 0.1571 \end{cases}$$

$$\Rightarrow \quad \begin{matrix} f_n = 0.25\ Hz \\ \zeta = 0.1 \end{matrix}$$

6.6 2nd Application conclusions

Using one of these three simple methods it is therefore possible to obtain a fairly realistic information about the dynamic characteristics of natural frequency f_n and damping ratio ζ of each periodic dynamic mode.

This information can also be used to predict or not the use of a particular flight control system necessary to make these values acceptable, i.e. in such a way that they comply with both the Regulations and the requirements.

7 Conclusions

In the two previous applications two different C++ codes were therefore used: these codes must then be subjected to appropriate validation and certification tests in accordance with the current regulations in the software field.

Finally, it will be necessary to decide which hardware to use to view the windows created by the software.

There are two main possibilities:

- use an external screen properly connected with the Odroid;
- use a computer screen, which in turn replaces the Odroid.

In fact, there may be the need to use a computer instead of the Odroid if the computing power of the latter is not sufficient to perform the various data acquisition and processing cycles with an appropriate frequency.

However, both choices are very valid: the laptop allows to use a single device to perform calculations and to display information. The Odroid and an external monitor, instead, allow, respectively, to perform the various calculations and display the information.

An external monitor that could be used is the Beetronics Full-HD Monitor 15'':

Figure 58: Beetronics Full-HD Monitor 15'' [30]

The Odroid can be connected to the monitor using an HDMI cable.

The new Flight Test Instrumentation (FTI) allows both to have a greater quantity of information than the previous one, and to carry out various operations in real time directly using the graphic interface created. It is also easier and more intuitive to use.

References

[1] EASA, April 2018. [Online]. Available: https://www.easa.europa.eu/sites/default/files/dfu/FTOM%20Guide.pdf.

[2] «Wikipedia,» [Online]. Available: https://en.wikipedia.org/wiki/Flight_test.

[3] EASA, [Online]. Available: https://www.easa.europa.eu/sites/default/files/dfu/Annex%20to%20EDD%202015-026-R.pdf.

[4] EASA, 4 September 2013. [Online]. Available: https://www.easa.europa.eu/sites/default/files/dfu/Presentation%204%20-%20Flight%20test%20organisation.pdf.

[5] PCB Piezotronic, "General Signal Conditioning Guide".

[6] https://en.wikipedia.org/wiki/Software_development_kit.

[7] Brüel & Kjær, 20 11 2013. [Online]. Available: http://pcfarina.eng.unipr.it/Public/Standing-Wave/bruel_2013-11-20.pdf.

[8] Hard Kernel, [Online]. Available: https://www.hardkernel.com/shop/odroid-n2-with-4gbyte-ram/.

[9] Matrix Vision, [Online]. Available: https://www.matrix-vision.com/serie-di-telecamere-industriali-compatte-usb3-vision-mvbluefox3-m2.html.

[10] Advanced Navigation, [Online]. Available: https://www.advancednavigation.com/product/spatial.

[11] D. L. Buglio, «La trasformata veloce di Fourier (FFT): analisi e implementazione in C++,» Milano, 2015.

[12] «Wolfram MathWorld,» [Online]. Available:

2] https://mathworld.wolfram.com/Danielson-LanczosLemma.html.

[1 [Online]. Available: https://en.wikipedia.org/wiki/Stability_derivatives.
3]

[1 O. S. N. Ranjan Ganguli. [Online]. Available:
4] https://www.researchgate.net/publication/28603906_Rotorcraft_Parameter_Identification_from_Real_Time_Flight_Data.

[1 [Online]. Available: https://en.wikipedia.org/wiki/Six_degrees_of_freedom.
5]

[1 [Online]. Available: https://en.wikipedia.org/wiki/Helicopter_rotor.
6]

[1 NASA, 1973. [Online]. Available:
7] https://www.nasa.gov/centers/dryden/pdf/87847main_H-806.pdf.

[1 L. C. S. G. Ronaldo Vieira Cruz. [Online]. Available:
8] http://downloads.hindawi.com/journals/mpe/2010/231594.pdf.

[1 «Rotor & Wind International,» [Online]. Available:
9] https://www.rotorandwing.com/2018/01/12/ray-prouty-archives-parameter-identification/.

[2 J. S. Bendat and A. G. Piersol, Measurement and Analysis of Random Data, John
0] Wiley & Sons, New York, 2000.

[2 [Online]. Available:
1] https://www.researchgate.net/publication/260944785_Applications_of_Parameter_Estimation_Methods_in_Helicopter_Identification.

[2 M. P. A. Q. Giorgio Guglieri, Meccanica del volo dell'elicottero, Milano: Società
2] Editrice Esculapio, 2018.

[2 A. Navigation. [Online]. Available:
3] https://www.advancednavigation.com/product/spatial.

[2 Advanced Navigation, [Online]. Available:

4] https://www.advancednavigation.com/sites/default/files/product_documents/Spatial%20Reference%20Manual%20v4.4_0.pdf.

[2 «HWRELOAD,» [Online]. Available: https://forum.hwreload.it/threads/cos%C3%A8-
5] un-vrm-a-cosa-serve-come-funziona.2160/page-2.

[2 «Wikipedia,» [Online]. Available:
6] https://en.wikipedia.org/wiki/Logarithmic_decrement.

[2 «andrew.cmu.edu,» [Online]. Available: https://www.andrew.cmu.edu/course/24-
7] 352/Handouts/logdecrement.pdf.

[2 «MathWorks,» 13th September 2018. [Online]. Available:
8] https://it.mathworks.com/matlabcentral/answers/418825-damping-factor-and-natural-frequency-out-of-time-response-data.

[2 «MathWorks,» 8 July 2018. [Online]. Available:
9] https://it.mathworks.com/matlabcentral/answers/265375-damped-cosine-wave-fitting.

[3 Beetronics, [Online]. Available: https://www.beetronics.it/monitor-led-ips-15-
0] pollici-metallo.