FINAL DEGREE PROJECT

"PROPOSING A UNIQUE ALTITUDE/HEIGHT SERVICE FOR UNMANNED TRAFFIC MANAGEMENT"

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Telecomunicació
i Aeroespacial de Castelldefels

Eric Rios Naval

Supervisor: Cristina Barrado Muxi

Air Navigation Engineering

Universitat Politècnica de Catalunya | EETAC

Abstract

On this final degree project, it will be discussed the problems associated with integrating Unmanned Aircraft Systems (UAS), including Remotely-Piloted Aircraft Systems (RPAS), into the airspace.
The different problems as aircraft separation or UAS altimetry will be exposed and commented.

First, it is going to be discussed about a UAS ATM Common altitude system and also the main reference system to compute the aircraft altitudes. Showing the advantages and disadvantages of each system, GNSS, QNH, QFE, FL, and in which situations they are used. Also it is going to be introduced the Visual Flight Rules, specially for very low altitudes.
It would be seen the differences between Europe and North America, what makes more difficult to set up a new system.

In order to solve the problem, a server is going to be created so the altitude from many different UAS or RPAS could be compared although they are not using the same system. Real data from the scientific department of the US will be used on the server to compute the changes of systems.
Once it's explained how the server is working, and which is the way the aircrafts or drones connects to it, the results will be shown and commented.

When the results are displayed, it would be possible to understand why is difficult to compare an UAS and a RPAS when both of them are using different altitude systems and how dangerous it could be.
The results will show real situations simulated over real airports, using the data from de US department. Then we will comment if the results obtained are the ones we were waiting and if new problems appear.

Index

1. Introduction

The purpose of this project is to solve the problems associated with integrating Unmanned Aircraft System (UAS), often called “Drones”, including Remotely-Piloted Aircraft Systems (RPAS), with all other aircrafts flying on the air.

The use of drones, or Unmanned Aircraft Systems, has become increasingly popular in recent years as technology advances and pricing have made them more accessible to individuals. It has been used for personal use but also commercial functions.

One of the main problem is that currently there are three acknowledged methods of determining the altitude of an aircraft, and drones and VFR flights are not using the same. The conflict appears when a drone and a VFR could be flying both at 1,000 m but from a different reference. One of them could be flying 1,000 m above the Mean Sea Level (MSL) and the other could be flying 1,000 m above the terrain. When this problem appears, it is necessary to convert one of these altitudes to the other, so it is possible to know the real separation between them and avoid dangerous situations.

The aim is to build a server that consists on transferring the data that provides the drone (altitude, latitude and longitude) into a measure system that compares it with VFR flights, IFR flights and vice versa.

The main part of this project will consists on a Python 3 code. We will create a server for several costumers, specially for UAS and RPAS.
The function of this server is to be prepared so when the client, VFR or drone, asks to change its altitude system, the server returns its altitude into QNH, QFE or GNSS.
So when it’s computed, it can be compared with others VFR in order to avoid collisions.

2. UAS ATM Common Altitude

This project is based in a topic validated in a Workshop organised at EUROCONTROL Brussels in April 2017. The purpose of that workshop was to discuss the problems associated with integrating Unmanned Aircraft System (UAS), colloquially called “Drones”, and included remotely, Piloted Aircraft Systems (RPAS), into the airspace that also includes many manned flights.

The many activities for which unmanned aircraft systems (UAS) are used, from military through commercial to leisure, can lead to their sharing airspace with conventional aircraft. In order to be maintained a separation between all users of this airspace, it is essential that the altitudes of all of these aircraft should be known unambiguously.

The three distinct points are:

- Flights rules
- Airspace, or operating environment, assessment
- A common altitude reference system

This last point is where this project is based, to compute a common altitude reference system with all UAS, RPAS and manned flights, in order to avoid conflicts between them.

2.1. The main problem

There are currently four acknowledged methods of determining the altitude of an aircraft. All of them are calculated using a pressure difference with respect to a known data point, using standard equipment:

- QFE – height above the local airport
- QNH – altitude above a given reference mean sea level (MSL)
- Flight level (FL) – Indicator of altitude within the International Standard Atmosphere (ISA), where the atmospheric pressure at MSL is defined 1013.25 hPa : 1FL = 100 ft.
- GNSS altitude

These methods will be discussed later on the chapter 3.

Whereas convectional manned aviation uses pressure altitude obtained from barometric readings, UAS often use other systems such as satellite-derived altitudes (GNSS). While each of these different systems can enable safe separation on its own, with some buildings or with the terrain, they can each have different altitude values from each other. A common altitude reference system needs to be established.

Also it is important to know, that there are many advantages and disadvantages, economic and technical, to all of the altitude measurement systems available.

3. Altitude measurements

In aviation, altitude can be computed by several systems, on this chapter we will comment the most important and used measurements. As it has been commented before, drones and VFR are using different measurements system to compute its altitude.

First of all, it is necessary to introduce the Q code. Q code is a standardized collection of three-letter codes all of which start with the letter "Q". It is an operating signal initially developed for commercial radiotelegraph communication and later adopted by other radio services.

- **QNH**

This measurement mode is a Q code indicating the atmospheric pressure adjusted to mean sea level. It is used by pilots, ATC (Air Traffic Control), and low frequency weather beacons to refer to the barometric setting which, when set on an aircraft's altimeter, will cause the altimeter to read altitude above mean sea level. An airport QNH will cause the altimeter to show airport altitude, that is, the altitude of the centre point of the main runway above sea level on landing, irrespective of the temperature.

In North America, this altitude is given in hundredths of inches of mercury, but in most parts of the world, QNH is given in hectopascals.

$$1\ hPa = 0.001\ bar$$
$$1\ hPa = 100\ Pa$$

- **QFE**

This system refers to the altimeter setting that will cause the altimeter to read the height above a specific aerodrome or ground level, and therefore read zero on landing.

ATC (Air Traffic Control) will pass the relative pressure changes to pilots on clearing them to descend below the transition level, as a part of ATC clearance, on request of the pilot or when the pressure changes.

- **FL (Flight Level)**

In aviation, a flight level is defined as a surface of constant atmosphere pressure which is related to a specific datum, nominally expressed in hundreds of feet. The altitude is computed assuming an International standard sea-level pressure datum of 1013.25 hPa, what means that is not necessarily the same as the aircraft's real altitude. The surfaces are separated from other surfaces by specific pressure intervals. The Flight Level number is always a multiple of 500ft, therefore always ending in 0 or 5.

e.g. FL310 = 31,000 feet above mean sea level when the pressure at sea level is 1,013.2 mbar.

When talking about the Flight Level, it is also necessary to talk about how changes the temperature while increasing or decreasing the altitude.

The atmosphere can be split into layers depending on whether temperature is increasing or decreasing with increasing altitude.

This graph shows how temperature changes with altitude, what means how changes due to the distance above sea level.

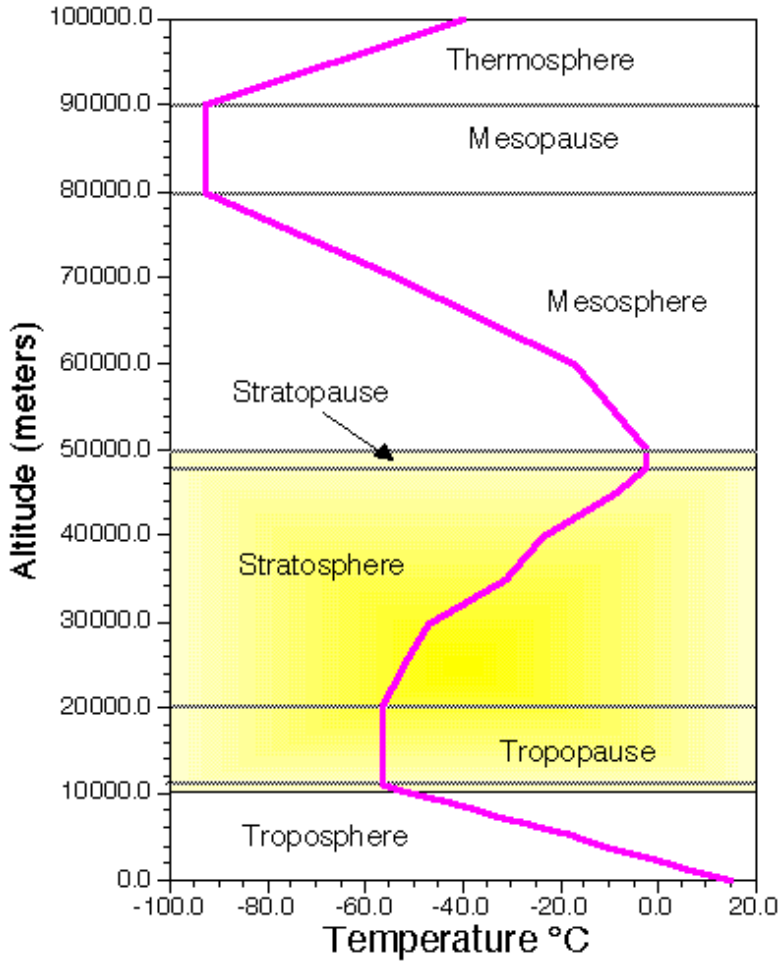

Figure [1] – Temperature vs Pressure

In figure [1] it is shown how the temperature increases or decreases depending on the air layer.
Until the troposphere (< 12,000 m), the temperature decreases 6.5 ºC per kilometre.
It means 1 ºC per 154 m of altitude. The troposphere contains most of the water vapour in the atmosphere and is where most of the clouds and weather occurs.

Temperature remains almost constant between 10 and 20 Km and then increases with increasing altitude between 20 and 50 Km. The stratosphere is a very stable air layer while the troposphere can be stable or unstable. The term of increasing temperature with increasing altitude is called an inversion, what makes the stratosphere so stable.

To explain the increasing temperature with increasing altitude process in the stratosphere it's necessary to talk about the ozone layer. The absorption of ultraviolet light by ozone warms the air in stratosphere and explains why the air can increase its temperature.

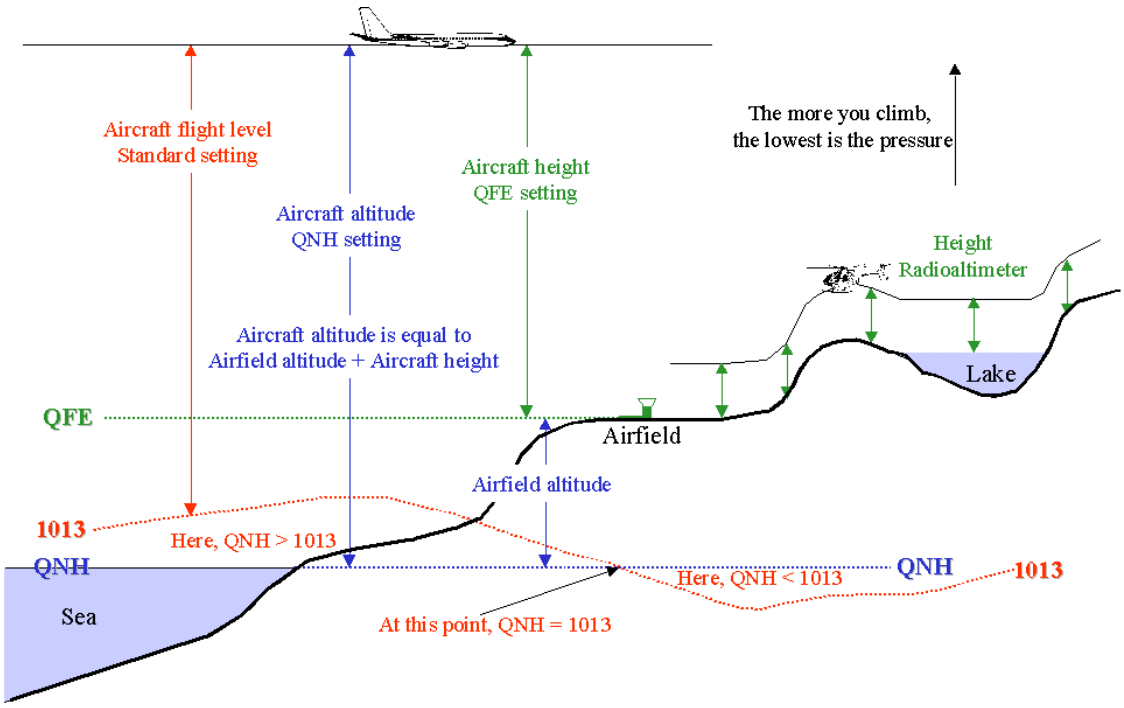

Figure [2] – Different altitude measurements

On figure [2] we can see graphically, the different ways in which the altitude could be computed. In red colour we can see the aircraft Flight Level.

Only above the transition level, which depends on the local QNH, are flight levels used to indicate altitude. As commented in other points, below the transition level feet are used.

The Radiometer Height, expressed in colour green, it is the real altitude above terrain immediately below the aircraft. Radio altimeters determined altitude by measuring the time between transmission of a radio signal from the aircraft and reception of the reflected signal. This equipment is expensive and not generally used.

- **TA (Transition Altitude)**

Transition Altitude is the altitude at or below which vertical position of an aircraft is controlled by reference to altitudes. When the aircraft is operating below the TA, altimeters are usually set to show the altitude measured above sea level. Above the TA, the aircraft altimeter pressure setting is normally adjusted to the standard pressure setting of 1013.25 hPa, so aircraft will be expressed as a FL (Flight Level).

The Transition Level is the lowest flight level available for use above the transition altitude. So the airspace between the transition altitude and the Transition Level is called Transition Layer.

QNH (in millibars)	Transition altitude (in feet)				
	3,000	4,000	5,000	6,000	18,000
1032–1050	FL25	FL35	FL45	FL55	FL175
1014–1031	FL30	FL40	FL50	FL60	FL180
996–1013	FL35	FL45	FL55	FL65	FL185
978–995	FL40	FL50	FL60	FL70	FL190
960–977	FL45	FL55	FL65	FL75	FL195
943–959	FL50	FL60	FL70	FL80	FL200

Figure [3] – Table for determining transition level

As it is seen on figure [3], Transition Altitude will vary depending on the QNH pressure.

In order to understand the figure above, let talk about an example.

e.g. if your QNH from the departure airport is between 996 to 1013 mbar, and the transition altitude is fixed at 5,000 ft, your first Flight Level would be FL55.

In Europe, the transition altitude varies from airport to airport and in many situations can be as low as 3,000 ft (910 m).

In the United States and Canada, the TA is fixed at 18,000 ft (5,500 m), and the airspace above is known as the Standard Pressure Region.

There are discussions to standardize the transition altitude within the Eurocontrol area *[1]*.

- Europe

In Europe and much of the rest of the world, the transition altitude varies from airport to airport as commented before. As the US case, it is a fixed value and is published on the documentation provided by the airport.

The normal barometric pressure setting procedure is a little different to that in North America.
The procedure that the pilots follow is the next one: climbing and cleared to a FL and set Standard Pressure Setting (1013 mbar). When descending is the reverse operation, clear to an altitude and set to the local QNH.
This procedure is done irrespective of how far above or below the TL/TA you are at the time.

According to ICAO, the Transition Level shall be located at least 300m (1,000ft) above the Transition Altitude, to let the Transition Altitude and the Transition Level to be used concurrently in cruising flight, with vertical separation ensured.

- North America

Under conditions of QNH at or above 1013 mbar, FL180 becomes the lowest useable FL. If the pressure is lower than 1013 mbar, the lowest useable FL becomes FL190 or even FL200. The restriction ensures that a minimum 1,000ft of vertical separation is maintained between the aircraft at 17,000ft on QNH.

The barometric pressure setting change shall be made in the standard pressure region, what means above 18,000ft on general cases. The change must take place just after entering or just prior to leaving the standard pressure region. What will take the pilots change to standard pressure (1,013.2 hPa) as they climb through 18,000ft.
Descending, even when cleared to an altitude at the time cruising level is vacated, the altimeters will remain on standard pressure until just prior to the transition level.

It is important to keep in mind that the Transition Level differs based on the atmospheric pressure at sea level (QNH), the lower the QNH, the higher transition level.

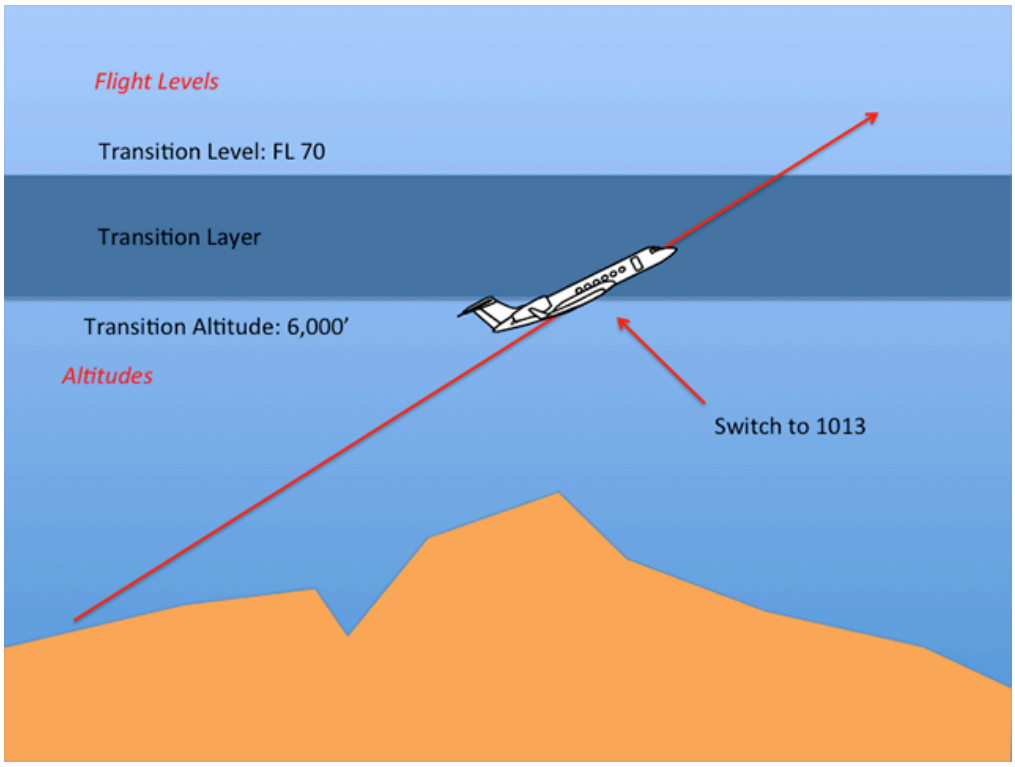

Figure [4] – Determining Transition Altitude

- **GNSS Altitude (GPS)**

Due to the Earth does not have a geometrically perfect shape, it is used the geoid to describe its unique and irregular shape and to approximate the mean sea level. However, only recently more substantial irregularities appeared in the surface. These irregularities are and order of magnitude greater than experts had predicted.

The shape of the geoid was calculated based on the hypothetical equipotential gravitational surface. There is a significant difference between this mathematical model and the real object. However, even the most mathematically sophisticated geoid can only approximate the real shape of the earth.

Mean Sea Level is defined as the zero elevation for a local area. The zero surface referenced by elevation is called a vertical datum. The MSL surface is in a state of gravitational equilibrium. It can be regarded as extending under the continents and is a close approximation of the geoid.

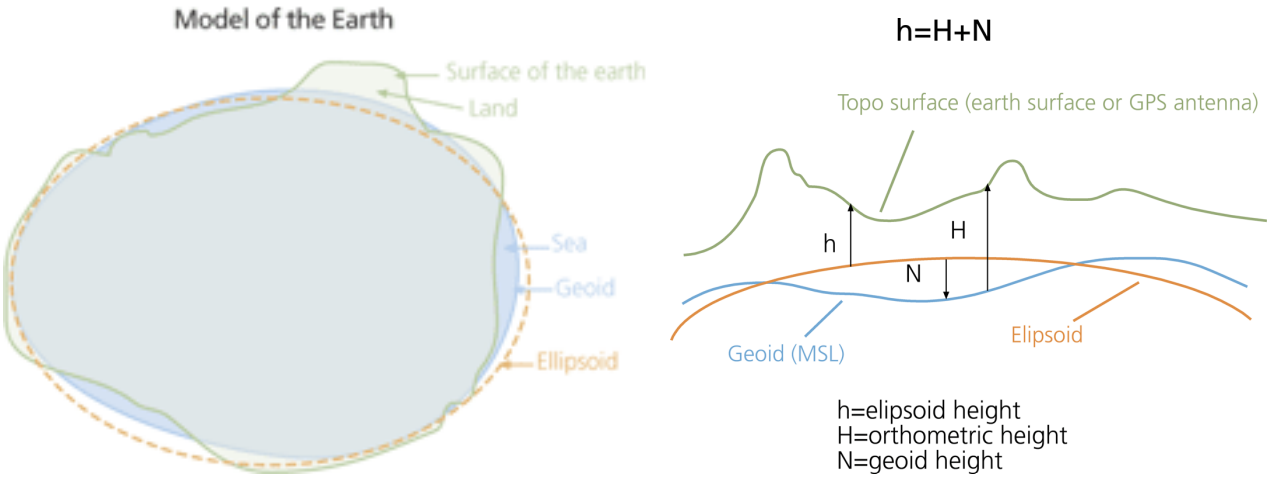

Figure [5] – Model of the Earth

A satellite navigation system uses satellites to provide autonomous geo-spatial positioning. It allows small electronic receivers to determine their location, with high precision, using time signals transmitted along a line of sight by radio from satellites.

Global coverage for each system is generally achieved by a satellite constellation of 18-30 medium Earth orbit (MEO) satellites spread between several orbital planes. The actual systems vary, but use orbital inclinations of > 50 º and orbital periods of roughly twelve hours, at an altitude of about 20,000 Km.

The accuracy of GPS height measurements depends on several factors but the most crucial one is the "imperfection" of the earth's shape. The GPS uses height h, shown on the figure before. This is the height above the reference ellipsoid that approximates the earth's surface. GPS also uses an ellipsoid coordinate system for both its horizontal and vertical datum.

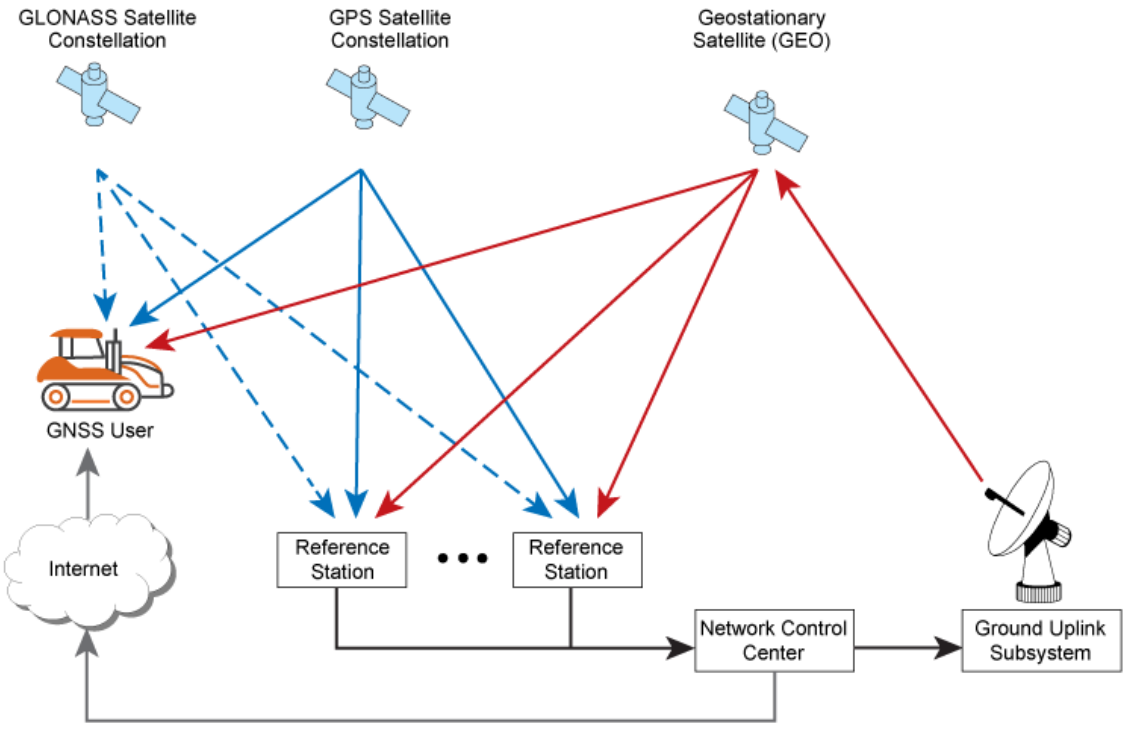

Figure [6] – GNSS system

4. Visual Flight Rules (VFR)

Visual flight rules are a set of regulations under which a pilot operates an aircraft in weather conditions (VMC) generally clear enough to allow the pilot to see where aircraft is going. VFR require a pilot to be able to see outside the cockpit, to control the aircraft's altitude, navigate and avoid obstacles and other aircraft or drones.

Because of the limited communication and navigation equipment required for VFR flight, a VFR aircraft may be subject to limitations if and when it is permitted in controlled airspace.

- **Low flying rules in the EU**

In all EU Members states, the Standardises European Rules of the Air (SERA) *[2]* apply; these set out a minimum altitude of 500ft above any obstacle within a radius of 500ft. Even thought, many countries apply their own rules, for instance in the UK, the "500ft Rule" allows pilots to fly below 500ft as long as they are no closer to any person, vessel, vehicle, building or structure.

- **Low flying rules in the US**

In the United States, Part 91 of the Federal Aviation Regulations controls the minimum safe altitudes by which aircraft can be operated in the National Airspace System. There is an over reaching general requirement to maintain sufficient altitude that in the fateful case a power unit fails, and emergency landing without undue hazard to persons or property on the surface can be made.

5. National Centers for Environmental Information

Several categories of model data are available through NOAA's (National Operation Model Archive). These broad categories of date are Numerical Weather Prediction, Climate Prediction, Reanalysis and Derived/Other Model Data.

For our project, we will be using IGRA data (Integrated Global Radiosonde Archive), the second version.

The IGRA consists of quality-controlled radiosonde and over 2,700 pilot balloon observations of temperature, humidity, pressure and wind at stations across all continents. There is data recorded since 1905, and the data are updated on a daily basis.

In its final form, IGRA is the largest and most comprehensive dataset of quality-assured radiosonde observations freely available. Its temporal and spatial coverage is most complete over the United States, western Europe, Russia and Australia.

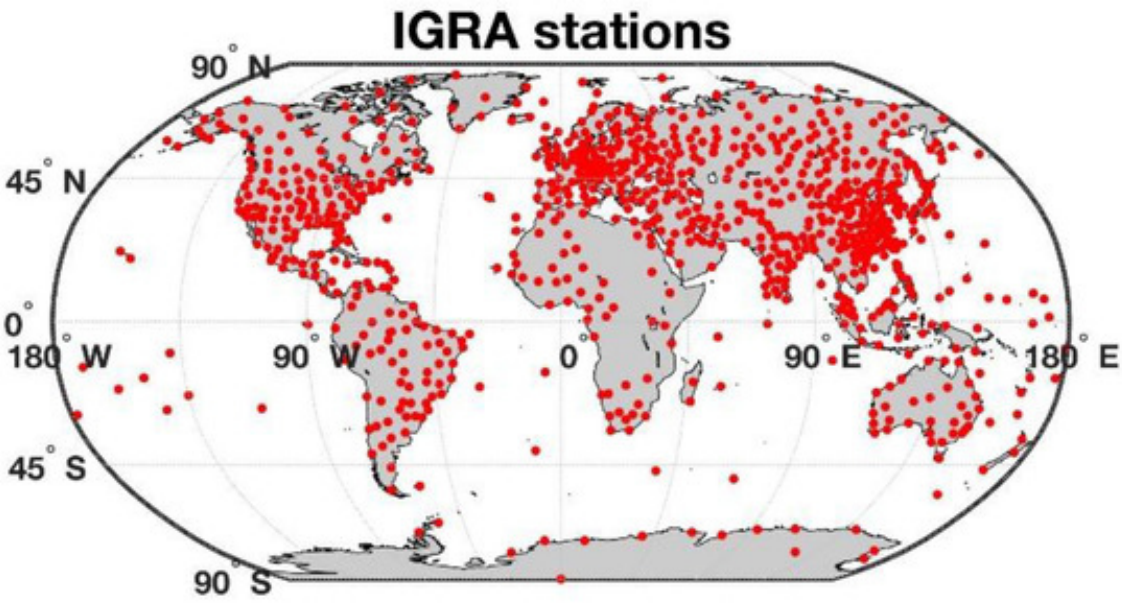

Figure [7] – IGRA stations

IGRA consists of three components:

- Individual soundings
- Monthly means
- Sounding-derived parameters

The one that we need is the individual soundings, that is organized into one file per station, this will provide us important information: temperature and pressure in a specific altitude. That is what we need to compute the real QNH pressure of the drone at a certain altitude.

First, we need to compare the latitude, longitude and altitude of the drone, and see which stations fit better with its position. Then we will search into the file of these stations and find the latest data recorded at this point.

5.1. Data from NOAA's

On the Python code, we will be using data from *https://www1.ncdc.noaa.gov/pub/data/igra/* to compute de real altitude that the drone is flying. First of all, we will need to select from the station list, the ones that are interesting for our study.

- **Station List**

This document is a ".txt" file with the information of all the stations that have recorded data in the last years.
This would be the style of the station list:

```
ACM00078861  17.1170  -61.7830   10.0    COOLIDGE FIELD (UA)             1947 1993  13896
AEM00041217  24.4333   54.6500   16.0    ABU DHABI INTERNATIONAL AIRPOR  1983 2018  35987
AEXUAE05467  25.2500   55.3700    4.0    SHARJAH                         1935 1942   2477
AFM00040911  36.7000   67.2000  378.0    MAZAR-I-SHARIF                  2010 2014   2179
AFM00040913  36.6667   68.9167  433.0    KUNDUZ                          2010 2013   4540
AFM00040938  34.2170   62.2170  977.0    HERAT                           1978 1988   1107
AFM00040948  34.5500   69.2167 1791.0    KABUL AIRPORT                   1961 2018  17524
AFM00040990  31.5000   65.8500 1010.0    KANDAHAR AIRPORT                1976 2014   5934
AGM00060355  36.8830    6.9000    3.0    SKIKDA                          1974 1976    639
AGM00060360  36.8330    7.8170    4.0    ANNABA                          1973 2008  30088
AGM00060390  36.6833    3.2167   25.0    DAR-EL-BEIDA                    1948 2018  68224
AGM00060402  36.7170    5.0670    6.0    BEJAIA-AEROPORT                 1973 1988  12372
AGM00060419  36.2830    6.6170  694.0    CONSTANTINE                     1973 2008  21262
AGM00060425  36.2170    1.3330  141.0    CHLEF                           1973 1977   3213
AGM00060430  36.3000    2.2330  715.0    MILIANA                         1973 1990  14988
AGM00060445  36.1830    5.2500 1040.0    SETIF                           1983 1989   5598
```

Figure [8] – Extract of Station List .txt

where the first column is the station ID, then the latitude, longitude and elevation, station code (only for US, Puerto Rico and Virgin Islands), name of the station, first and last year of record in the sounding data, and the last column is the number of soundings in the sounding data record.

The data that most matters us are ID of the station, latitude, longitude, elevation above the geoid and the last year of record in the sounding data.

Now is when we need to choose the stations that will take part of our database, that will be these ones that have data reordered on the last year. Once the stations are selected, we can begin to study the necessary data.

- **Data List**

This data will be also stored in a ".txt" file document that are updated once a day in the early morning Eastern Time (US). The latest observations usually become available within two calendar days of when they were taken. That's the reason why in the code we will compare the necessary data with the data recorded 2 or 3 days before. Also it is possible to have real-time data, but that's a payment option available on the website.

Data files are available for two different time spans:

- In subdirectory *data-por,* that contain the full period of record
- In subdirectory *data-y2d*, files only contain soundings from the current and previous year. This would be the files that we will be using, because for our simulation, as it is commented before, we will be using data from the previous 2-3 days, what means we do not care about the recorded data of the years before.

Each file in the *data-por* and *data-y2d* subdirectories contains the sounding data for one station. The name of the file corresponds to a station's IGRA 2 identifier. So, in the code, we will find the nearest station to our drone, search the code, and then download the document from the website.
For example, if the drone is 26º latitude, 78º altitude and an elevation of 200m, the nearest station is at Gwalior and the code is INM00042361. So the file to search in the website is *INM00042361-data-beg2019.txt.zip*.

Each sounding consists of one header record and n data records, where n, that is given in the header record, is the number of levels in the sounding. It's the last number before the ncdc code.

```
#SAM00041112 2018 01 01 00 0011   176 ncdc-gts ncdc-gts  182333   426500
21 -9999   79500B-9999     90B-9999    120   210     31
20 -9999   79400 -9999     94B-9999    160   295     10
20 -9999   79300 -9999    136B-9999    250 -9999 -9999
20 -9999   76700 -9999    148B-9999    260 -9999 -9999
10 -9999   70000  3151B     90B-9999    270   230     15
20 -9999   69100 -9999     80B-9999    240 -9999 -9999
20 -9999   68100 -9999 -9999 -9999 -9999   270     57
20 -9999   65400 -9999 -9999 -9999 -9999   275     51
20 -9999   64100 -9999     56B-9999    380 -9999 -9999
20 -9999   61900 -9999 -9999 -9999 -9999   180     51
20 -9999   58100 -9999 -9999 -9999 -9999   185     21
20 -9999   56100 -9999 -9999 -9999 -9999   185     51
20 -9999   55300 -9999 -9999 -9999 -9999   200     57
20 -9999   54700 -9999    -47B-9999    240 -9999 -9999
20 -9999   54400 -9999 -9999 -9999 -9999   175     67
20 -9999   53700 -9999    -41B-9999    430 -9999 -9999
20 -9999   50100 -9999 -9999 -9999 -9999   180    134
10 -9999   50000  5850B   -81B-9999    290   180    134
```

Figure [9] – Extract of data recorded document

The figure above is an example of one of the documents commented before. The first line is the header that consists on the identification code, the date of the data, the observation hour of the sounding (in UTC on the date indicated in the YEAR/MONTH/DAY fields) and the real release time of the sounding in UTC. The next column is an important value, the number of levels in the sounding, what means the number of data records that follow (i.e., 35 means 35 lines of recorded data).
The following codes are the data source code for non-pressure levels in the sounding, which are the levels that have not recorded the data clearly. The last two columns are the latitude and longitude, which also appears on the station list.

When a "-9999" value appears, means that the data recorded was not a quality data and these lines of data are not useful. So it is necessary to compute an approximation if we need the data from this altitude.

The code "ncdc" refers to the data that is recorded from National Climatic Data Center, that is form the U.S. Department of Commerce, in Washington DC.

Once the header is clear, it's time to understand how is displayed the recorded data.

We will put interest in bytes from 10-15 that is the reported pressure in Pa, the 17-21 bytes that is the reported geoid height, and the 23-27 bytes that is the reported temperature on ºC to tenths (e.g., 11 = 1.1 ªC).

The python code will consist on many functions, that will compute from the altitude displayed on the drone the real pressure on this point to compute the QNH or QFE, vice versa.

6. Project development

Now it's time to talk about how we would need to solve all the problems commented before between the Unmanned Aircraft System and the Piloted Aircraft Systems. On the first part, it would be explained how the code is working and which information is used for it. Later, on the next chapter, the results would be shown with real situation cases.

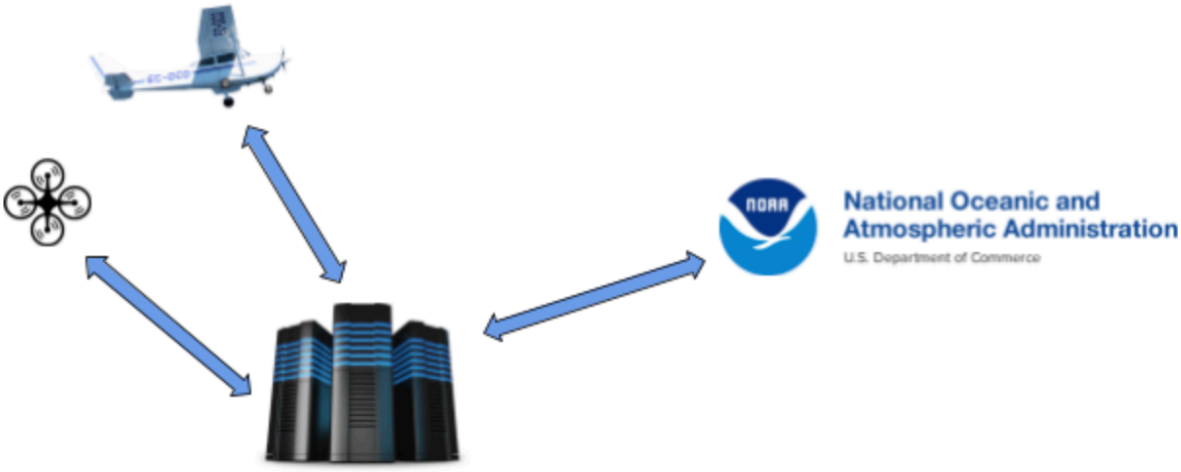

Figure [10] – Project architecture

On figure [10], it is shown the basic architecture of the project. Always is the client, drone or VFR flight, who starts the communication with the server. Then, when the server has all the information needed, connects to NOAA's database and downloads the necessary data. Once the calculation is computed, the server sends the information asked to the client and show the results.

6.1. Solution

This part of the report, consists on the explanation about the code we have performed. As it is talked before, the Python code consists on a server prepared to compute altitudes from different systems to the systems available, that would be QNH, GNSS and QFE.

We will have two separated codes, but running together. One part would be the client, that is the simulation of the drone or the VFR flight, and the other one will be the server.
First of all, when the client wants to interact with the server, it would be asked to its ID, and the type of aircraft (drone or VFR flight). Then the server would ask about the kind of information that the client needs, if want to compute its altitude into QNH, QFE or GNSS.

- **Drone**

The drone, could transfer its altitude from GNSS to QNH or QFE, depending on what its needed. On both cases, the drone would be asked about its latitude, longitude and its altitude above the geoid. Then, the drone would need to respond to the system about in which system wants its altitude transferred, in case that it wants the QFE, would be also necessary to ask the altitude of the reference airport. Once it sends the position, it would be compare with all the IGRA stations.
When we have the ID code of the station, it would be necessary to download from *https://www1.ncdc.noaa.gov/pub/data/igra/* the latest data recorded for these points.

Once the server looks for the pressure for its altitude on the NOAA's data, it will use the barometric formula and compute the result asked.

- **VFR flight**

On this second case, these aircrafts can also convert its altitude into the other ones. As well as the drone, it would be asked first about the actual altitude system, and then to which altitude system wants to be transferred. In all of the cases, would be asked to its altitude, never the pressure, because in a real case, the pilot would only see the altitude represented on its altimeter. Also, if the pilot wants to convert its altitude from QFE, or wants to be transferred into QFE, it would be asked about the reference airport elevation.

Once, the server has all this information, would start computing the final result. First

If the VFR flight only wants to compute its altitude between QFE and QNH, it won't be necessary to connect with IGRA stations, its only one calculation.

If the client wants the GPS altitude, the server will compare the aircraft's position with (latitude and longitude) with all the IGRA stations, and find which is the nearest one. Then the altitude will be changed to pressure, using the barometric formula, so it can be compared with the NOAA's data.

As the case before, when we have the ID code of the station, it would be necessary to download data from NOAA, and search the latest data recorded for these points.

Then, the server would compare the data from all the altitudes recorded on this station and select the altitude that fits with its pressure.

7. Validation

On this part, is where we will put all the possible situations under test in our server and explain the results obtained. All the figures shown on this chapter are results from the Python code server created, that can be found on the annexes.

The average time of the request, since the client asks to the server until the server shows the results is less than 15 seconds.

- **Test 1**

A drone (UAS) flying at 3,150 m (GPS), over the Denver International Airport (KDEN).

The airport is located in (39.8617, -104.673) and its AMSL (metres above sea level) is 1,655 m.

QFE altitude

```
Are you a drone (1) or a VFR (2)? 1
What's your ID? D001
Which measurement do you want? (1=QNH, 2=QFE) 2
Which is the altitude of the reference airport? (In meters) 1655
What's your altitude (m)? 3150
What's your latitude? (-90 to 90) 39.8617
What's your longitude? (-180 to 180) -104.6731
Client> D001
Client> 2
Client> 3150
Client> 39.8617
Client> -104.6731
Client> 1655
Server< Hello D001, your GPS altitude is: 3150.0 m over the sea level || Your QFE
  altitude is: 1302.28 m over the reference airport.
```

QNH altitude

```
Are you a drone (1) or a VFR (2)? 1
What's your ID? D001
Which measurement do you want? (1=QNH, 2=QFE) 1
What's your altitude (m)? 3150
What's your latitude? (-90 to 90) 39.8617
What's your longitude? (-180 to 180) -104.6731
Client> D001
Client> 1
Client> 3150
Client> 39.8617
Client> -104.6731
Server< Hello D001, your GPS altitude is: 3150.0 m over the sea level || Your QNH
  altitude is: 2957.28 m over the sea level.
```

This figure shows graphically the position of the drone, on the situation tested before with all the results obtained.

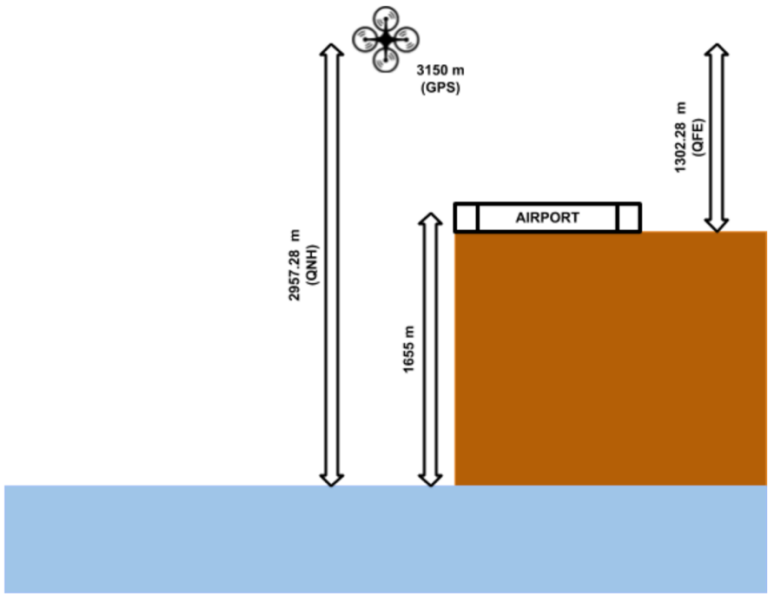

Figure [11] – Test 1

- **Test 2**

A drone (UAS) flying at 2,000 m (GPS), over Zaragoza, Spain. Its position is (41.6561, -0.8777) and wants to compare its altitude with a VFR flight, flying at 1,700 m (QNH).

QNH altitude

```
Are you a drone (1) or a VFR (2)? 1
What's your ID? D002
Which measurement do you want? (1=QNH, 2=QFE) 1
What's your altitude (m)? 2000
What's your latitude? (-90 to 90) 41.6561
What's your longitude? (-180 to 180) -0.8777
Client> D002
Client> 1
Client> 2000
Client> 41.6561
Client> -0.8777
Server< Hello D002, your GPS altitude is: 2000.0 m over the sea level || Your QNH
  altitude is: 1922.04 m over the sea level.
```

The calculation proves that the real separation between the drone and the VFR flight is more than 200 m.

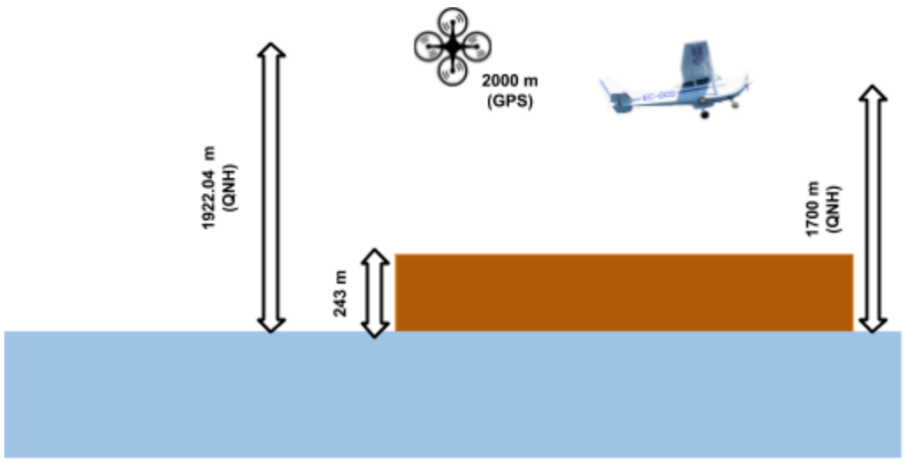

Figure [12] – Test 2

- **Test 3**

An aircraft (VFR) flying at 1,300 m (QNE), over the Charlotte Douglas International Airport (KCLT). The aircraft wants to compare its altitude with a drone, flying at 2,300 m (GNSS).

The airport is located in (35.2139, -80.9431) and its AMSL (metres above sea level) is 228 m.

GPS altitude

```
Are you a drone (1) or a VFR (2)? 2
What's your ID? V003
Which measurement do you have? (1=QNH, 2=QFE) 2
Which is the altitude of reference the airport? (In meters) 228
What's your altitude (m)? 1300
What's your latitude? (-90 to 90) 35.2139
What's your longitude? (-180 to 180) -80.9431
Client> V003
Client> 2
Client> 1300
Client> 35.2139
Client> -80.9431
Client> 228
Server< Hello V003, your QFE altitude is: 1300.0 m over the reference airport || Your
  QNH altitude is: 1528.0 over the sea level || Your GPS altitude is: 1739.2 m over the
  sea level
```

The server shows that the aircraft is flying more than 500 m down the drone.

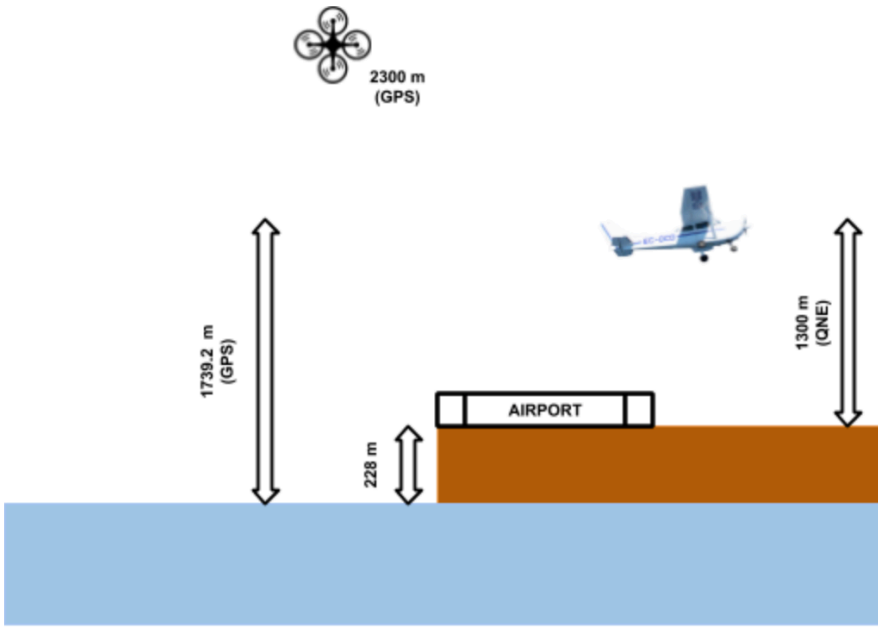

Figure [13] – Test 3

- **Test 4**

An aircraft (VFR) flying at 2,600 m (QNH), over Prague, Czech Republic. Its position is (50.0880, 14.4208) and wants to compare its altitude with a drone, flying at 2,100 m (GNSS).

```
Are you a drone (1) or a VFR (2)? 2
What's your ID? V004
Which measurement do you have? (1=QNH, 2=QFE) 1
What's your altitude (m)? 2600
What's your latitude? (-90 to 90) 50.0880
What's your longitude? (-180 to 180) 14.4208
Client> V004
Client> 1
Client> 2600
Client> 50.0880
Client> 14.4208
Server< Hello V004, your QNH altitude is: 2600.0 m over the sea level || Your GPS
  altitude is: 2565.41 m over the sea level.
```

As it's shown in the figure, the results prove that the VFR flight is more than 400 m over the drone.

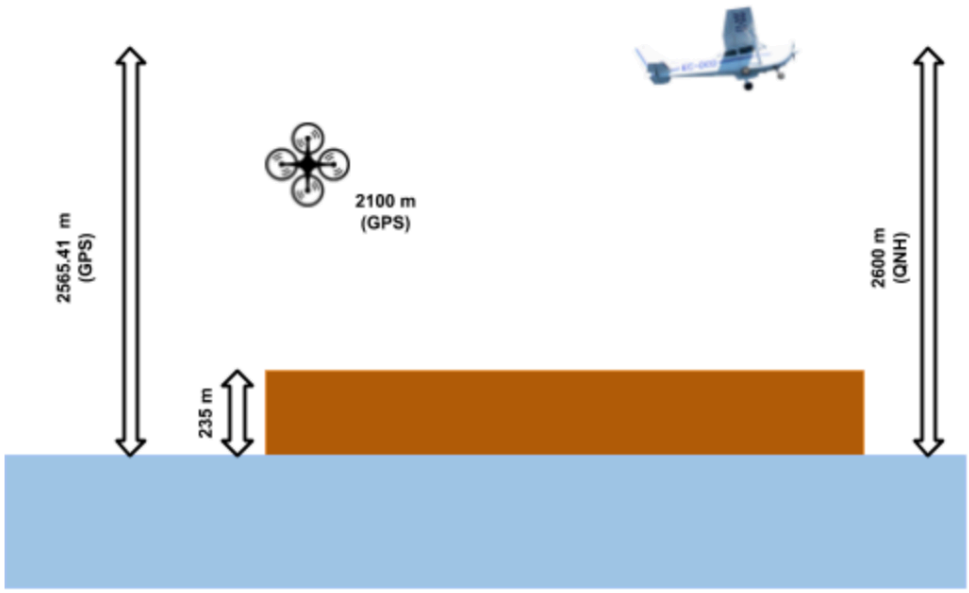

Figure [14] – Test 4

8. Conclusions

To sum up this project, I wanted to show that is possible, with all the test done before, to compute one altitude reference system for UAS and RPAS, but it is difficult to implement it.

Nowadays, there are lots of drones and aircrafts, and it is not easy to change its altitude system from all of them.
In a near future, it could be possible to implement a server connection to all the new drones and aircrafts delivered so they can change its altitude and compare to another one.
One solution could be, set a year in a near future, for example 10 years, and fix to all the UAS and RPAS to have all the necessary equipment to connect to a server that changes its altitude system.

The results show that depending on your position and altitude, the computation of the altitude between the difference systems may differ higher or lower.

As it is seen before, it could dangerous to be flying in a same region with different references, because in several situations the changes of measure are considerable. It is possible to create many different servers, with different accuracy's, the more money you could invest, the better accuracy you obtain. It may be possible to use real data-time and not to use data from 1 or 2 days before.

With the drones being part of our lives, its also important to refresh many rules from the VFR, in order to adapt them to new possible situations.

9. Bibliography

[1]: Public data of the Integrated Global Radiosonde Archive at *https://www1.ncdc.noaa.gov* (last access on 6-Feb-2019)

[2]: Altitude aeronautical measurements at *https://www.skybrary.aero* (last access on 6-Feb-2019)

[3]: Mean Sea Level, GPS, and the Geoid at *https://www.esri.com* (last access on 6-Feb-2019)

[4]: UAS ATM Common Altitude Reference System at *https://www.eurocontrol.int* (last access on 6-Feb-2019)

[5]: Overview oh the Integrated Global Radiosonde Archive at *https://journals.ametsoc.org/* (last access on 6-Feb-2019)

Articles:

[1]: "A Common European Transition Altitude; An ATC perspective", Eurocontrol on 26 Septmeber 2013

[2]: Commission Implementing Regulation (EU) 2016/1885 of July 2016 at *https://www.easa.europa.eu/document-library/regulations/commission-implementing-regulation-eu-20161185* (last access on 6-Feb-2019)

[3]: UAS ATM Common Altitude Reference System Guidelines by Peter Hullah at EASA

[4]: UAS ATM CARS Discussion Document by EUROCONTROL, DECMA and ACS

Figures:

[1]: Layers of the atmosphere based on temperature at *http://www.atmo.arizona.edu* (last access on 6-Feb-2019)

[2]: The three types of aircraft elevation at *https://www.aviationcv.com/aviation-blog* (last access on 6-Feb-2019)

[3]: Flight level at *https://en.wikipedia.org* (last access on 6-Feb-2019)

[4]: Transition Altitude / Layer / Level at *http://code7700.com* (last access on 6-Feb-2019)

[5]: Shape of the earth at *https://www.esri.com* (last access on 6-Feb-2019)

[6]: Precise Point Positioning (PPP) at *https://www.novatel.com* (last access on 6-Feb-2019)

[7]: An Algorithm for Atmospheric Temperature and Water Vapor Profile Estimation from ATMS Measurements Using a Random Forests Technique at *https://www.mdpi.com/* (last access on 6-Feb-2019)

[8] and [9]: Altitude, pressure and temperature data at *https://www.ncdc.noaa.gov* (last access on 6-Feb-2019)

[10]: Figure created showing the project architecture

[11], [12], [13] and [14]: Figures created with the results from the server

10. Annexes

Python code:

- **Client**

```
import asyncio
import websockets


type = input("Are you a drone (1) or a VFR (2)? ")

if int(type) != 1 and int(type) != 2:

  print("ERROR 404, SYSTEM NOT FOUND")

async def sendmessage(uri):
    async with websockets.connect(uri) as websocket:

      #### DRONES

      if int(type) == 1:

        ID = input("What's your ID? ")
        sys = input("Which measurement do you want? (1=QNH, 2=QFE) ")

        if(int(sys) == 2):
            airport = input("Which is the altitude of the reference airport? (In
meters) ")

        altitude = input("What's your altitude (m)? ")
        latitude = input("What's your latitude? (-90 to 90) ")
        longitude = input("What's your longitude? (-180 to 180) ")

        await websocket.send(type)
        await websocket.send(ID)
        print(f"Client> {ID}")
        await websocket.send(sys)
        print(f"Client> {sys}")
        await websocket.send(altitude)
        print(f"Client> {altitude}")
        await websocket.send(latitude)
        print(f"Client> {latitude}")
        await websocket.send(longitude)
        print(f"Client> {longitude}")
        if (int(sys) == 2):
            await websocket.send(airport)
            print(f"Client> {airport}")

        respuesta = await websocket.recv()

        print(f"Server< {respuesta}")



      #### VFR

      if int(type) == 2:

        ID = input("What's your ID? ")
        sys = input("Which measurement do you have? (1=QNH, 2=QFE) ")
```

```
            if (int(sys)==2):
              airport = input("Which is the altitude of reference the airport? (In meters)
    ")

            altitude = input("What's your altitude (m)? ")
            latitude = input("What's your latitude? (-90 to 90) ")
            longitude = input("What's your longitude? (-180 to 180) ")


            await websocket.send(type)
            await websocket.send(ID)
            print(f"Client> {ID}")
            await websocket.send(sys)
            print(f"Client> {sys}")
            await websocket.send(altitude)
            print(f"Client> {altitude}")
            await websocket.send(latitude)
            print(f"Client> {latitude}")
            await websocket.send(longitude)
            print(f"Client> {longitude}")

            if (int(sys) == 2):
              await websocket.send(airport)
              print(f"Client> {airport}")

            respuesta = await websocket.recv()

            print(f"Server< {respuesta}")

    asyncio.get_event_loop().run_until_complete(
        sendmessage('ws://localhost:8766'))
```

- **Server**

```
import asyncio
import websockets
import math
import os
from read_stations import *
from read_info import *
from download_file import *
from read_METAR import *
from geographiclib.geodesic import Geodesic

R = 8.3144598
M = 0.0289644
g = 9.80665
Po = 1.013 * 10**5

stations = reading_stations()

a=1

async def echo(websocket, path):

    type = await websocket.recv()
    ID = await websocket.recv() # recibe informacion de cliente
    sys = await websocket.recv()

    if int(type) == 1:
        altitude_drone = float(await websocket.recv())
        ap = altitude_drone
```

```
    if int(type) == 2:
        altitude_VFR = float(await websocket.recv())
        ap = altitude_VFR

    latitude = float(await websocket.recv())
    longitude = float(await websocket.recv())

    if int(sys) == 2:
        altitude_Airport = float(await (websocket.recv()))


    print(f"Client< {ID}") # escribe la informacion
    print(f"Client< {sys}")
    print(f"Client< {ap}")
    print(f"Client< {latitude}")
    print(f"Client< {longitude}")


    encontrado = 0
    i = 0


    arc = Geodesic.WGS84.Inverse(stations[0].lat, stations[0].lon, latitude, longitude)
    mindist = arc['s12']
    i_min = 0

    while(i < len(stations)):

        arc = Geodesic.WGS84.Inverse(stations[i].lat, stations[i].lon, latitude,
longitude)
        dist_sta = arc['s12']

        print(dist_sta)

        if dist_sta < mindist:
            mindist = dist_sta
            i_min = i

        i = i + 1


    ID_station = str(stations[i_min].ID)

    print(ID_station)

    downloading_file(ID_station)
    [temp, press, h] = reading_P_T(ID_station)

    os.remove('' + ID_station + '.txt')


###ESTA PARTE SOLO PARA VFR !!!!!!

    if int(type) == 2:

        if int(sys) == 1:

            sys_1 = 'QNH'

            pressure = math.exp(altitude_VFR/(-8005))*100000

            print(pressure)
            print(press)

            i = 0
            encontrado = 0
```

```
            while (i < len(press) and encontrado == 0):

                if pressure < press[i] and pressure > press[i+1]:

                    altitude = ((pressure - press[i]) / (press[i+1] - press[i])) *
(h[i+1] - h[i]) + h[i]
                    encontrado = 1

                i = i + 1

                print(i)

            print(press)
            print(h)

            respuesta = f"Hello {ID}, your QNH altitude is: {altitude_VFR} m over the sea
level || Your GPS altitude is: {round(altitude, 2)} m over the sea level."


        if int(sys) == 2:

            sys_2 = 'QFE'

            ## NO USADO

            ## Calcular QNH local

            #
            # print(airport[0])
            #
            # P_ref = reading_METAR(airport)
            #
            # pressure_total = P_ref - pressure  # qne = aeropuerto - real
            #
            # while (i < len(press) and encontrado == 0): #Calculo altitud del mar,
tomando presion respecto aeropuerto
            #
            #     if pressure_total > press[i] and pressure_total < press[i - 1]:
            #         altitude = ((pressure_total - press[i]) / (press[i + 1] -
press[i])) * (h[i + 1] - h[i]) + h[i]
            #         encontrado = 1
            #
            #     i = i + 1
            #
            # i = 0
            # encontrado = 0
            # print(press)
            # print(P_ref)

            altitude_total = altitude_Airport + altitude_VFR

            pressure = math.exp(altitude_total / (-8005)) * 100000

            i = 0
            encontrado = 0

            while (i < len(press) and encontrado == 0):

                if pressure < press[i] and pressure > press[i + 1]:
                    altitude = ((pressure - press[i]) / (press[i + 1] - press[i])) * (h[i
+ 1] - h[i]) + h[i]
                    encontrado = 1

                i = i + 1
```

```
            respuesta = f"Hello {ID}, your QFE altitude is: {altitude_VFR} m over the
reference airport || Your QNH altitude is: {altitude_total} over the sea level || Your
GPS altitude is: {round(altitude, 2)} m over the sea level"

        print(f"Server> {respuesta}")

        await websocket.send(respuesta)


###ESTA PARTE SOLO PARA DRONES !!!!!!

    if int(type) == 1:

        # GPS a QNH

        # Buscamos el valor de temperatura REAL que corresponde a esta altitud

        i = 0
        encontrado = 0

        #ecuacion de la recta : T = (altitude - h[i])/(h[i+1] - h[i]) * (T[i+1] - T[i]) +
T[i]
        while encontrado == 0:

            if altitude_drone < h[0]:

                T = (altitude_drone - h[0]) / (h[1] - (h[0])) * (temp[1]-temp[0]) +
temp[0]
                P = (altitude_drone - h[0]) / (h[1] - (h[0])) * (press[1] - press[0]) +
press[0]


                encontrado = 1

            if altitude_drone > h[len(h) - 1]:

                last = len(h) - 1

                T = (altitude_drone - h[last-1]) / (h[last] - h[last-1]) * (T[last + 1] -
T[last]) + T[last]
                P = (altitude_drone - h[last - 1]) / (h[last] - h[last - 1]) *
(press[last + 1] - press[last]) + press[last]

                encontrado = 1

            if altitude_drone > (h[i]) and altitude_drone < (h[i+1]) and altitude_drone <
h[len(h) - 1]:

                T = ((altitude_drone - h[i]) / (h[i+1] - h[i])) * (temp[i+1]-temp[i]) +
temp[i]
                P = ((altitude_drone - h[i]) / (h[i + 1] - h[i])) * (press[i + 1] -
press[i]) + press[i]

                encontrado = 1

            i = i + 1

        if int(sys) == 1:

            sys_1 = 'QNH'

            print(P)
            print(press)
```

```
            print(h)

            altitude_QNH = (-8005) * math.log(P/100000)
            #altitude_QNH = ((-1)*(R*T)/(M*g)) * math.log(P/Po)

            respuesta = f"Hello {ID}, your GPS altitude is: {altitude_drone} m over the
sea level || Your QNH altitude is: {round(altitude_QNH, 2)} m over the sea level."


        # GPS A QNE

        if int(sys) == 2:

            sys_2 = 'QFE'

            print(P)

            altitude_QNH = (-8005) * math.log(P / 100000)
            altitude_QFE = altitude_QNH - altitude_Airport

            respuesta = f"Hello {ID}, your GPS altitude is: {altitude_drone} m over the
sea level || Your QFE altitude is: {round(altitude_QFE, 2)} m over the reference
airport."


        print(f"Server> {respuesta}")

        await websocket.send(respuesta)


asyncio.get_event_loop().run_until_complete(
    websockets.serve(echo, 'localhost', 8766))
asyncio.get_event_loop().run_forever()
```

- **Functions**

- downloading_file

```
import urllib3
import certifi
import zipfile
import io

def downloading_file(param):

    http = urllib3.PoolManager(cert_reqs='CERT_REQUIRED', ca_certs=certifi.where())
    url = 'https://www1.ncdc.noaa.gov/pub/data/igra/data/data-y2d/'+param+'-data-
beg2018.txt.zip'
    resp =http.request('GET',
                       url,
                       preload_content=False)

    zf = resp.data

    file_stream = io.BytesIO(zf)
    zipfile2 = zipfile.ZipFile(file_stream)


    name = zipfile2.infolist()
    info = zipfile2.open(name[0])
    a = info.read()

    f = open(param+".txt", "w")
    f.writelines(a.decode("utf-8"))
    f.close()
```

- read_info

```
-   import datetime


    def reading_P_T(param):



        f = open(param+".txt")
        temp=[]
        press=[]
        h=[]
        i=1
        a=0
        encontrado = 0

        ahora = datetime.datetime.now()
        year_today = int(ahora.year)
        month_today = int(ahora.month)

        if ahora.day > 2:
           day_today = int(ahora.day) - 2
        if ahora.day < 3:
           day_today = 30
           month_today = month_today - 1

        print(ahora.year, ahora.month, ahora.day)

        # Buscamos la ultima captura del sensor, la que mas se acerque al dia de hoy
        while(encontrado == 0):

          if f.readline(1) == '#':
            borrar = f.readline(12)
            year = f.readline(4)
            borrar = f.readline(1)
            month = f.readline(2)
            borrar = f.readline(1)
            day = f.readline(2)
            borrar = f.readline(9)
            levels =f.readline(4)

            #print(day, month)
            #print(levels)



          if int(year) == year_today and int(month) == month_today and int(day) <=
    day_today + 2:
             encontrado = 1

          if encontrado == 0:
             z=1
             while(z <= int(levels) + 1): #Recorremos toda la info de un dia que no nos
    interesa
               a=f.readline()
               z = z + 1



        # Nos quedamos con las muestras de presion y temperatura

        borrar = f.readline()
        i = 1

        while(i <= int(levels)):
```

```
            borrar=f.readline(9)
            press.append(f.readline(6)) #En Pa
            borrar=f.readline(1)
            h.append(f.readline(5)) #En m
            borrar = f.readline(1)
            temp.append(f.readline(5)) #Se divide entre 10 porqué esta en deci
    centigrados y pasamos a Kelvin

            borrar = f.readline()

            i = i + 1



        a = len(temp)
        temp.pop(a-1)
        press.pop(a-1)
        a = len(temp)

        # Borramos los datos erroneos
        i = 0
        while(i < a):


            if temp[i] == '-9999' or press[i] == '-9999' or h[i] == '-9999':
                temp.pop(i)
                press.pop(i)
                h.pop(i)

                a = len(temp)
                i = i - 1

            i = i + 1

        i = 0
        while (i < len(temp)):

             temp[i] = int(temp[i]) / 10 + 273
             press[i] = int(press[i])
             h[i] = int(h[i])
             i = i + 1

        h.pop(i)
        return temp, press, h
```

- read_stations

```
class stru:
    def __init__(self, id='', lat='',lon='',ele=''):
        self.ID = id
        self.lat = lat
        self.lon = lon
        self.ele = ele

def reading_stations():
    stations = []

    f = open("station-list.txt")
    data = [line.split() for line in f.readlines()]

    a = len(data)
```

```
    i=0
    f = open("station-list.txt")
    while i < a :


        estacio=stru(data[i][0],float(data[i][1]),float(data[i][2]),float(data[i][3]))

        borrar = f.readline(77)
        lastyear = f.readline(4)
        borrar = f.readline()

        if int(lastyear)== 2018:
          stations.append(estacio)

        i = i + 1

    stations_1 = stations

    return stations_1
```

- read_METAR (not used)

```
import requests


def reading_METAR(airport):


    link =
'http://www.universalweather.com/regusers/publictools/corp_comp/metar_taf_notam.html?icao
=' + airport + ''


    response = requests.get(link)
    data = response.text

    i = 0
    encontrado = 0
    while encontrado == 0:

        if data[i : i+12] == 'OBSERVATIONS':

            encontrado = 1

        i = i + 1

    encontrado = 0

    print(data)

    i = i+13

    if airport[0] == 'K':

        while encontrado == 0:

            if data[i] == 'A':
                pressure_airport = data[i+1 : i+5]

                encontrado = 1
            i = i + 1
```

```
            print(i)

        pressure_airport = int(pressure_airport) * 33.8639

    else:

        while encontrado == 0:

            if data[i] == 'Q':

                pressure_airport = data[i+1 : i+5]

                encontrado = 1
            i = i + 1

        pressure_airport = int(pressure_airport) * 100


    return pressure_airport
```