UCLouvain

École polytechnique de Louvain

CubeSats in formation for Earth observation

Author: **Laurent Paucot**
Supervisor: **Christophe Craeye**
Readers: **Maxime Drouguet**, **Sebastien Le Maistre**, **Jan Thoemel**
Academic year 2022–2023
Master [120] in Electrical Engineering

Acknowledgements

First I would like to express my gratitude to my supervisor, Professor Christophe Craeye. His huge knowledge and expertise, his communicative passion, his positivity and advice were precious values throughout this journey.

I would like to address particular thanks to Ir. Maxime Drouguet. I am sincerely grateful for his availability, his presence and his patience over the past few months. A large part of this master thesis wouldn't have been possible without his permanent implication. Working alongside him was also very enlightening.

I am grateful to the other members of the jury, Dr. Sebastien Le Maistre and Dr. Jan Thoemel for the time they have agreed to devote to the reading of my thesis.

Thanks should also go to my parents for their constant support and for proofreading.

Lastly I would like to mention Brieuc for his unwavering presence in the library during the writing of this thesis.

Abstract

This thesis investigates the use of CubeSats in antenna array synthesis for satellite-based Earth observation, with a particular focus on GNSS reflectometry applications. It explores CubeSats as an innovative solution to achieve highly directional antennas while reducing mechanical and cost constraints.

The study identifies suitable CubeSat configurations for antenna array synthesis and defines quantitative targets for resolution and sidelobe levels near the specular point. A methodology using spiral configurations is proposed and the trade-off between resolution and sidelobe levels is explored. It is shown that such configurations have continuous aperture-like behavior near the main beam and that the size of this zone is predictable.

In addition, the thesis addresses challenges related to digital beamforming and radar localization in the glistening zone for Earth imagery production. It discusses synchronization challenges caused by the inability to transmit the local oscillator through a cable and proposes a solution using a master CubeSat that transmits its local oscillator to the rest of the swarm. Three experiments are performed to characterize the system, showing satisfactory estimation accuracy while identifying synchronization problems with variable time spacing between CubeSats.

Keywords: satellite-based Earth observation, CubeSats, GNSS-R, digital beamforming, radar localization, wireless synchronization.

Contents

List of Figures

List of Tables

1 Introduction

Satellite Earth observation refers to the collection and processing of data imaging the Earth's features using payloads mounted on satellites. In recent years, the sector has seen a significant increase in the number of companies involved. A wide variety of business or industrial sectors, such as agriculture, energy, insurance or security, rely on Earth observation to adopt more sustainable practices motivated by the need to reduce greenhouse gas emissions. Earth observation is also essential for monitoring the state and evolution of the environment during extreme weather events, the frequency of which is unfortunately expected to increase [1], [2].

Earth observation using reflected GNSS signals has been developing since the end of the 1990s. Its main advantage is that it does not require the transmission of high-power RF signals from the same satellite. Since the discovery of the interest that these reflected signals could represent, several campaigns have been carried out using passive antennas. The so formed bi-static radars allow to carry out sensing and monitoring of various phenomena [3]. Such applications imply the ability to focus on very narrow regions on the Earth surface, requiring to have very directive antennas. To date, large aperture antennas are used, which implies the launch of size-adapted satellites, causing cost and mechanical problems. As a solution, it has been suggested to replace continuous aperture antennas with an irregular antenna array using CubeSats in formation.

If antenna arrays are widely used on Earth, it is a completely new and innovative idea to rely on them to perform satellite-based Earth observation. However, there are still some technical challenges to be overcome, and this thesis is part of the research and development of solutions to those challenges.

More specifically, the purpose of this work is to study CubeSat formations that allow one to partially reproduce the behavior of an equivalent continuous aperture antenna. In fact, the low number of available CubeSats, due to cost constraints, makes the approximation of an equivalent continuous aperture antenna difficult. Furthermore, to image a small region of the Earth, one must first be able to perform source separation and localization. From this point of view, it is necessary to synchronize the elements of an array, which is usually done by a wired transmission of the local oscillator of the demodulator. This thesis therefore also proposes the characterization of a first prototype receiver array, including wireless synchronization.

This thesis is organized as follows. In Section 2, the context and the project in which

this work takes place will be discussed. GNSS-R technology will be briefly described and the use of CubeSats in formation will be motivated. Section 3 will propose a study of antenna array layouts by defining quantitative objectives, detailing the methodology, and proposing the interpretation of results based on the concept of space tapering. Section 4 will present a first characterization of a bi-static radar composed of two CubeSat prototypes performing angle-of-arrival estimation. This section starts with a presentation of the prototype and continues with a description of the digital beamforming algorithm as well as the synchronization management of the CubeSats. A methodology for the estimation of the measurement accuracy is then presented. The section ends with an interpretation of the results. Section 5 will conclude this thesis by recalling the objectives, summarizing the results and opening the way for further work and development on this project.

2 Context

2.1 GNSS-R

A Global Navigation Satellite System is defined by the EU Agency for the Space Program as "a constellation of satellites providing signals from space that transmit positioning and timing data to GNSS receivers" [4]. These data are generally used to determine a position, a velocity or to provide timing information both on Earth and in space. The main existing GNSS systems are

- The Global Positioning System (GPS) managed by the USA
- GLONASS provided by the Russian Federation
- The European Union's GALILEO constellation
- China's BeiDou navigation satellite system

These navigation systems operate in several frequency bands, the most common one being the L1 band. Since the frequency carriers must be multiples of 10.23 MHz, the L1 carrier is equal to

$$f_{L1} = 154 \cdot 10.23 = 1575.42 \text{ MHz} \tag{1}$$

which will be considered as the GNSS operating frequency for the remainder of this thesis.

It quickly became apparent that, in addition to the applications mentioned above, the GNSS signal could also be used to sense the Earth's atmosphere. While all of the above applications use the direct GNSS signal, the idea of using the signals reflected from the Earth was born. This method, known as GNSS reflectometry (GNSS-R), is particularly useful for altimetry or for remote sensing of various natural elements from ocean to land, including snow water content, sea or ice salinity and soil moisture analysis [3].

The peculiarity of GNSS-R techniques is the concept of opportunity. While a traditional active monostatic radar is responsible for transmitting and receiving the (reflected) signals, GNSS-R applications operate in a bi- (or multi-) static configuration. This means that a passive radar, potentially far away from a GNSS swarm and not co-operating with it, can use the direct and reflected signals that are emitted for free from its point of view. A schematic [5] of a bi-static GNSS-R radar is shown in Figure 1.

As illustrated in Figure 1, GNSS transmitters and receivers are usually not on the same orbit. Let us remind that Low Earth Orbit (LEO) generally refers to

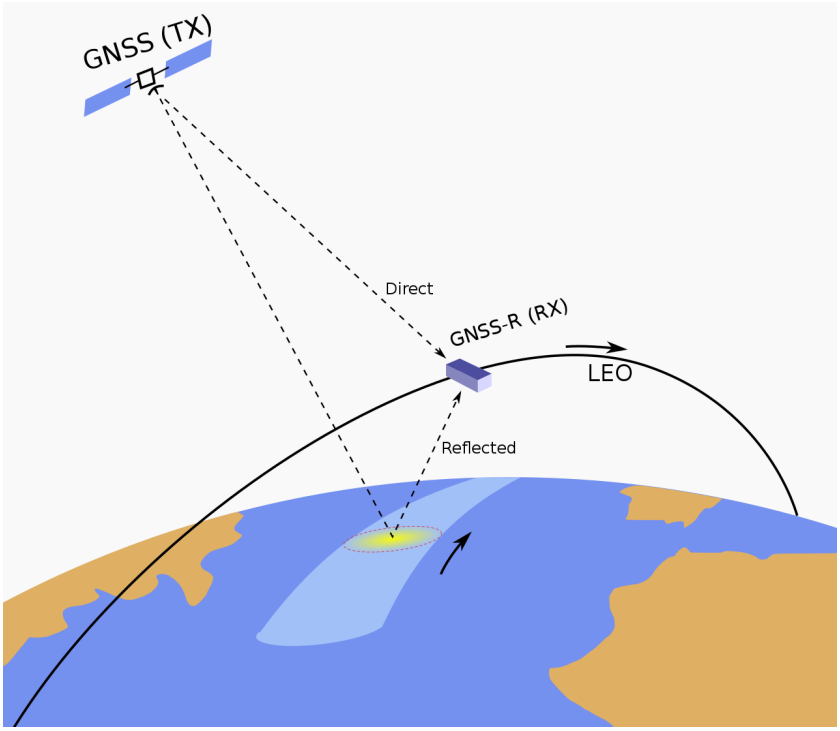


Figure 1: GNSS-R bistatic radar [5]

an altitude of less than 1000 km, while Medium Earth Orbit refers to an orbital period of less than 24 hours, corresponding to an altitude of 35 786 km [6].

A receiving satellite (RX) in a Low Earth Orbit (LEO) receives both the direct signal from a GNSS element (i.e. the transmitter TX) in a much higher orbit (usually Medium Earth Orbit) and the reflected signal on Earth. This bi-static radar can be turned, for example, into an altimeter by measuring the delay between the direct and reflected signals.

One of the main advantages of GNSS-R sensor technology is its low cost and low power consumption. In fact, these receiving satellites require mainly passive instrumentation, which allows the use of smaller equipment and thus the deployment of larger constellations of this type of satellite [7], [3].

2.2 Antenna array and CubeSats

In the context of GNSS-R Earth observation, there is an interest in being able to focus on quite narrow regions. Taking into account that the receiving satellite is

located at a distance of about 400 km (LEO) from the Earth, the antenna must have a high directivity as well as reasonable side lobes around the main lobe. There are two ways to increase the directivity:

- Increasing the electrical length of the antenna
- Combining several antennas into an array

The electrical length G is a dimensionless measure of a conductor :

$$G = \frac{l}{\lambda} \tag{2}$$

where l is the length of the conductor and λ is the wavelength [8]. This means that increasing the directivity would mean increasing the size of the antenna. The problems with a large antenna are the cost increase due to the larger satellite size required, as well as more potential mechanical problems related to its deployment.

The second solution can be considered by having a swarm of small satellites each supporting a single element. This constellation would thus form an antenna array. This solution allows one to deal with high directivity and to steer in order to focus on a pre-determined region on Earth.

The planned system suggests the use of CubeSats. These CubeSats are miniaturized satellites with a size of $10 \times 10 \times 10$ cm^3, which can be stacked. This class of satellites includes several benefits, such as

- Low development cost
- Short time development
- Low launching cost

The latter is the main advantage. Indeed, CubeSats can be launched as a secondary payload on another launch and benefit from a standard deployment system (e.g. California Polytechnic State University's P-POD) [9], [10].

In a GNSS-R application, the main limitation of such a device is the difficulty in designing a high-gain antenna. However, this is not a limitation in this case as, a multiplicity of those CubeSats will be launched to form an array.

If constraints on directivity and sidelobe levels need to be reached, the phased array technique can be used. This consists in applying well-chosen complex weights to each element to achieve a particular combination of signals. This in turn forms

a high-directivity beam in a pre-determined direction [11]. However, this technique, also known as power tapering, has the disadvantage of reducing the SNR (Signal to Noise Ratio). In fact, it involves deciding not to receive the signal at full power in some elements without reducing the noise level. Directivity and sidelobe level requirements should thus be reached by working on the CubeSat spacings.

3 Antenna layouts

3.1 Objectives

As explained in Section 2, the purpose is to create an antenna layout that would be directive enough to be able to observe relevant phenomena on Earth. This means that the main beam should be very narrow to have a high resolution. In addition, sidelobes should be avoided or minimized close to the main beam. Indeed, as explained in [12] the specular point is rather a specular zone called the *glistening zone*. This is mainly caused by the roughness of the Earth surface that induces a scattering of the electromagnetic waves around the specular point. In this region the presence of side-lobes could degrade the accuracy of the Earth observation.

In a quantitative way, the following objectives have been fixed. Knowing that

- The antenna array is 400 km far away from the Earth surface
- The glistening zone is defined as a disk with an approximate 10 km diameter
- The number of elements should not exceed 32 (for cost reasons)

the following specifications should be reached :

- A 1 km resolution
- Sidelobes of the array factor limited to -10 dB inside the glistening zone

No constraint is imposed on sidelobes out of the zone. The 400 km altitude (Low Earth Orbit) corresponds to the minimum one allowed for a CubeSat when designing antennas as recalled in [11]. The size of the glistening zone has been determined in [12] and corresponds to the accuracy that was possible to get in extreme cases, such as very mountainous regions.

3.2 System of coordinates

Before addressing the method employed to reach the objectives, the system of coordinates used in the rest of this section needs to be introduced. This system is shown in Figure 2.

It was decided to work on the basis of a system of spherical coordinates using the geographer's convention with

- ρ the distance from the origin to the point P
- θ the longitude

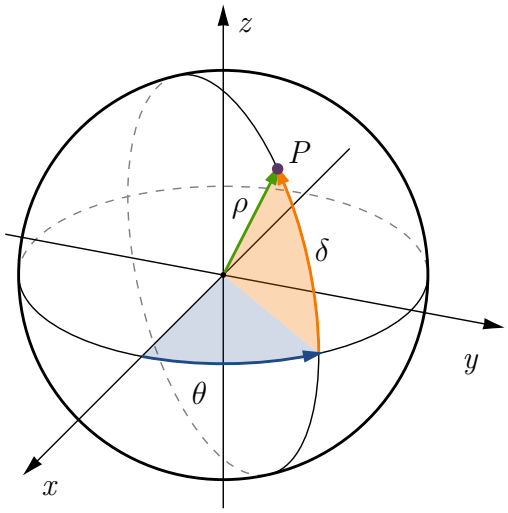


Figure 2: System of spherical coordinates [13]

- δ the latitude

The co-latitude $\varphi = 90° - \delta$ is also defined and will rather be used than the latitude.

The exchange from the Cartesian system to the spherical one is recalled in equations (3).

$$\begin{cases} x &= \rho \sin\varphi \cos\theta \\ y &= \rho \sin\varphi \sin\theta \\ z &= \rho \cos\varphi \end{cases} \tag{3}$$

It will be considered in this section that the planar antenna array will always lay in the X-Y plane.

3.3 Method

3.3.1 Radiation pattern and directivity

In order to characterize an antenna array, interest is turned to the study of its radiation pattern and by extension in its directivity. Under the following assumptions

- Similar antennas
- Isotropic antennas
- Individual complex amplitude $a_n = 1$

it is recalled that the radiation pattern, defined as "A graphical representation of the radiation properties of the antenna as a function of space coordinates" in [14] of an array composed by N elements is given by

$$\vec{F}(\vec{u}) = \sum_{n} \exp\left(jk\vec{\rho}_n \cdot \hat{u}\right) \tag{4}$$

where k is the wavenumber, $\vec{\rho}_n$ is the vector going from the origin of the local system of coordinates (generally placed at the center of the array) to the element n and $\hat{u}$ is an unitary vector representing the direction of observation.

The isotropic antennas hypothesis can be assumed taking into account the antenna configuration proposed in [15]. Indeed, the square arrangement of dipoles and parasitic elements allows to produce a radiation pattern with a sufficiently large main beam with regard to the size of the glistening zone. It will thus be considered that the radiation pattern of the array will mainly depend on its array factor with a negligible influence of the element pattern in the nadir direction.

The directivity, "In a given direction, 4π times the ratio of the radiation intensity in that direction to the total power radiated by the antenna"[14], can be expressed as a function of the radiation pattern :

$$D(\hat{u}) = 4\pi \frac{|F(\hat{u})|^2}{\iint |F(\hat{u})|^2 \, d\Omega} \tag{5}$$

In spherical coordinates, one gets

$$D(\theta, \varphi) = 4\pi \frac{|F(\theta, \varphi)|^2}{\int_0^{2\pi} \int_0^{\pi} |F(\theta', \varphi')|^2 \sin\theta' \, d\theta' d\varphi'} \tag{6}$$

An example of a directivity diagram is available in Figure 3. The width of the main lobe, 3 dB below the maximum directivity is used to compute the resolution by solving the right triangle given in Figure 4. One can note that the width of the main beam could evenly be computed on the array factor.

The resolution r can thus be computed like

$$r = d \tan\left(2\alpha\right) \tag{7}$$

where d is the distance between the center of the array and the Earth surface, equal to 400 km in this case.

If the glistening zone is assumed to be a 10 km diameter disk, the latter equation can be reversed to determine the angular region where sidelobes should be minimized.

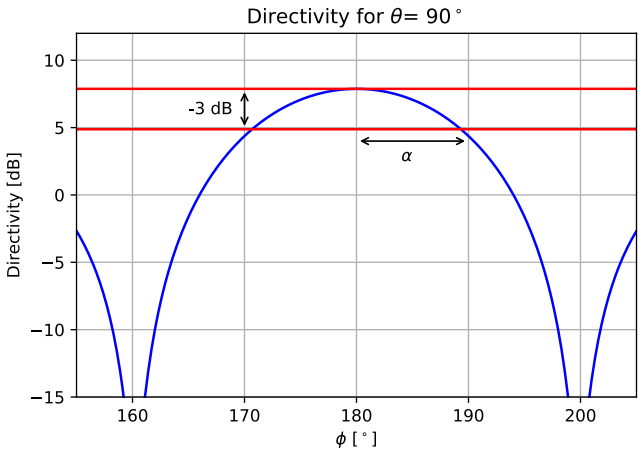


Figure 3: Beamwidth

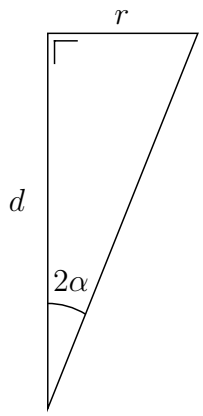


Figure 4: Resolution computation

3.3.2 Space tapering

As developed in [16], a relevant methodology when searching to achieve a certain beamwidth and given sidelobe levels is to synthesize non regular planar arrays to approach the radiation pattern of a continuous aperture that would have the same radius as the array. This technique called space tapering is thus a form of spatial sampling of a continuous aperture. Indeed the continuous integral over the disk can be approached by a summation considering a large number of elements. The radiation pattern can then be written as

$$\sum_{n=1}^{N-1} e^{jk\hat{u}\cdot\vec{\rho}} \approx C^{-1} \iint f(x,y) e^{jk\hat{u}\cdot\vec{\rho}} dxdy \tag{8}$$

where $f(x,y)$ is the scalar source distribution, $\vec{\rho}$ is the vector from the center of the array (resp. aperture) to each element (resp. point of the aperture) and C is a normalizing constant such that

$$C = \frac{1}{N} \iint f(x,y) dxdy \tag{9}$$

The sum can be seen as a Monte-Carlo integration assuming that the elements are distributed proportionally to the distribution $f(x,y)$. If a uniform source distribution

$$f(x,y) = 1 \tag{10}$$

is assumed, elements are expected to be homogeneously spread on the disk.

An estimate of the width of the main beam and of the sidelobes of an homogeneously sampled continuous aperture is also proposed in [16] :

$$\sin \alpha \approx \frac{\lambda}{2R} \tag{11}$$

with R the radius of the equivalent continuous aperture. Recalling equation (7) and that a 1 km resolution is expected, it comes

$$\sin\left(0.0716^\circ\right) = \frac{\lambda}{2R} \tag{12}$$

and $R \approx 80$ m knowing that $\lambda = 0.2$ m.

The antenna array should thus spatially sample (if possible with a uniform density) a 80 m radius disk.

An example of a continuous aperture radiation pattern is given in Figure 5.

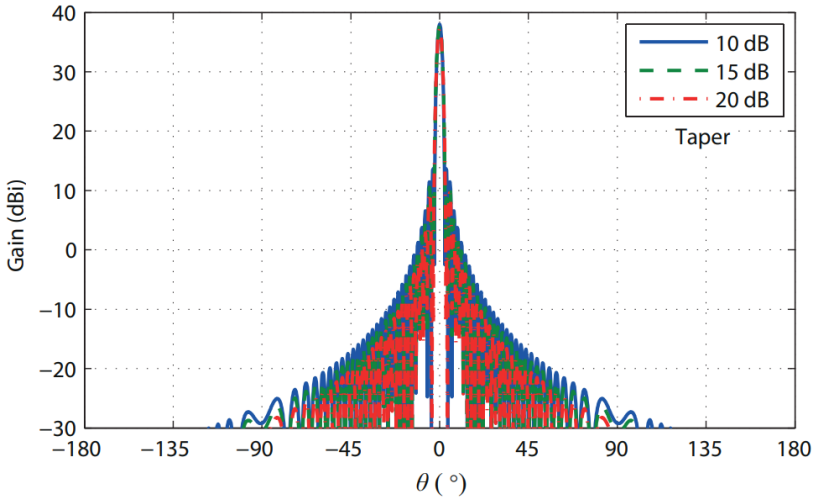


Figure 5: Gain of a paraboloidal reflector [17]

The gain in [dBi] is given for different illumination tapers. The pattern is characterized by a series of sidelobes with progressive decrease in the amplitude. This will serve as a reference in the results analysis in Section 3.4.

3.3.3 Spirals

It was determined in [12] that arranging elements along spirals would be a relevant way of research in order to reach the specifications given in Section 3.1. Indeed,

this sort of layout enables approaching a homogeneous density of elements over a given surface.

The idea is to create one "branch" and then to replicate it according to the number of desired arms. The problem can be formalized with equations (13) and (14).

$$\begin{cases} x_n &= x_{n-1} + \rho_n \cos\left(\phi_{n-1} - \psi_n\right) \\ y_n &= y_{n-1} + \rho_n \sin\left(\phi_{n-1} - \psi_n\right) \end{cases} \tag{13}$$

and

$$\phi_n = \phi_{n-1} + \psi_n \tag{14}$$

x_n and y_n are the absolute Cartesian coordinates of the element n along one branch. Those coordinates are relative to the element $n-1$ requiring x_0 and y_0 to be fixed. Elements do not need to be evenly spaced along a circle arc, so that for each interval a given distance ρ_n and curvature ϕ_n is associated. A schematic of one arm is proposed in Figure 6. The black dots represent single elements of the array. It can be verified that ϕ_n is equal to $\phi_{n-1} - \psi_n$.

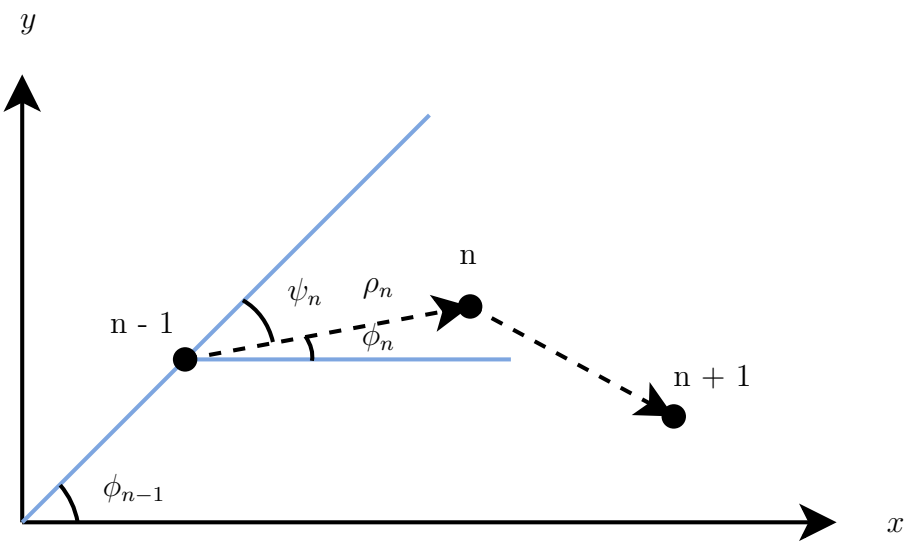


Figure 6: Arm of a spiral

Spirals are formed by combining several similar arms. Given a certain number of elements, the number of branches (or the number of elements by arm) needs to be chosen a priori or can also be an optimization parameter.

An example with 12 elements spread along 4 similar arms shifted by $360/4 = 90°$ is given in Figure 7. It can be noted that an element is present at the center of the layout. Indeed, for reasons related to the orbital mechanics, spirals will be rotating around this "master" CubeSat in the final application.

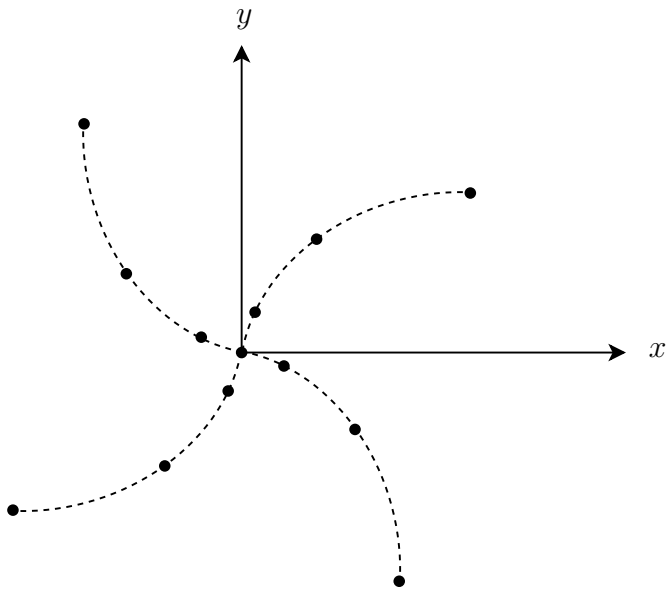


Figure 7: 4×3 elements spiral

In order to approach a uniform density, it is best to have a radial density roughly equal to the azimuthal density of elements. A way to achieve it is to progressively increase the spacing ρ along a given branch. If one decides to work with regular increments of ρ and ψ the following parameters need to be determined for an arm :

- $\rho_{\min} = \rho_0$: the first spacing
- $\rho_{\max} = \rho_{N-1}$: the last spacing
- $\psi_{\min} = \psi_0$: the first incremental rotation angle
- $\psi_{\max} = \psi_{N-1}$: the last incremental rotation angle

with $\phi_0 = \psi_0$. The intermediate values being evenly spread between minimum and maximum values.

3.4 Results

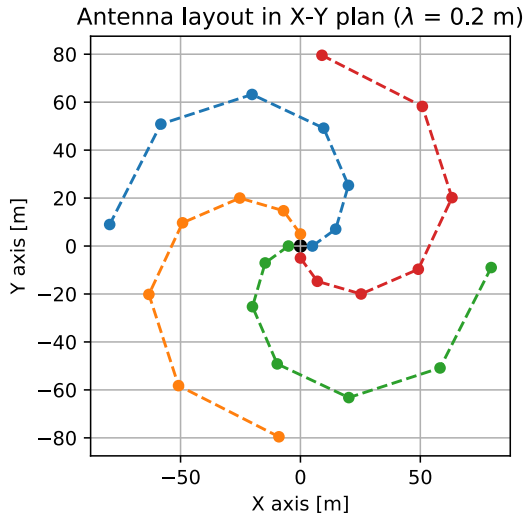
The main difficulty in the search for a configuration that would reach the requirements is the number of parameters and the large range of values that they can take.

It is thus required to try to fix some of them or at least to restrict the range of values. The first thing that was fixed is the number of elements and the number of arms. As working with more elements tends to yield better results (the integral in equation (8) being better approached with a high number of elements), the tested layouts will always be formed by 4 arms of 7 elements. The master CubeSat is then added at the center which gives a total of 29 antennas. Through a try-and-fail approach (so far), ranges of ρ and ψ providing good but not fully satisfactory results could be determined. The criteria are recalled to be a 1 km resolution and a minimization of the sidelobes in the glistening zone. Many configurations were then randomly tested in these ranges while keeping those giving the best results with regard to the objectives introduced previously.

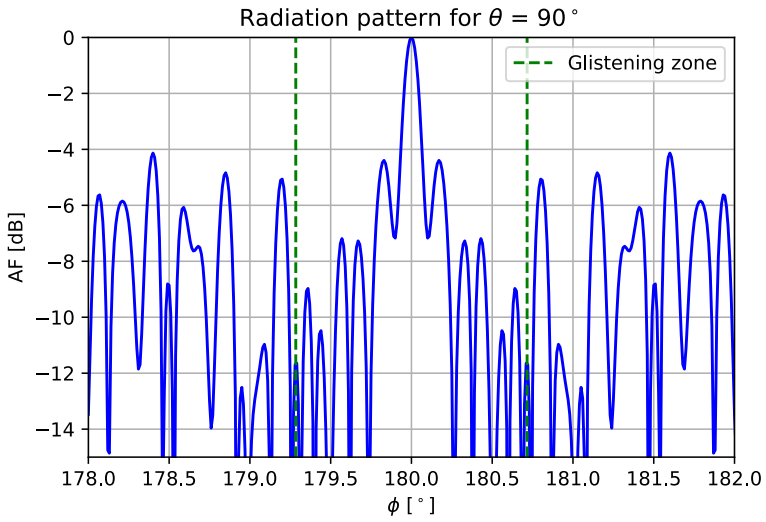
Recalling that the radius of the layout was estimated in Section 3.3.2 to be around 80 m if targeting a 1 km resolution, a first solution is proposed in Figure 8.

Figure 8(a) shows the antenna configuration. Each CubeSat is represented by a dot. Each spiral arm is represented by a different color and the master CubeSat is the black dot at the origin of the system. One can verify that the layout is inscribed in a 80 m radius circle. So far, no word has been said on the distance between elements. Recalling that the standard size of a CubeSat is $10 \times 10 \times 10$ cm^3, that the antenna system will have to be deployed (leading to a 30 cm side square) and that the position of each CubeSat will not be guaranteed to the nearest centimeter, a minimal distance of 1 m seems to be safe [15], [12]. In that case even the elements placed at the center of the system are sufficiently spaced as the smallest distance between two elements is equal to 5 m.

Figure 8(b) gives the radiation pattern obtained with the configuration proposed in Figure 8(a).It is recalled that the radiation pattern is assumed to be equivalent to the array factor as the pattern of an isotropic antenna has been assumed in Section 3.3.1. The array factor has been normalized and is shown in a decibel scale (vertical axis). The pattern is given in a cut corresponding to $\theta = 90°$ (see the system introduced in Figure 2) and in a range of 4° around the main lobe. The 10 km diameter glistening zone is represented by the green vertical dashed lines. The width of the main beam at -3 dB corresponds to a 0.79 km distance on the Earth surface which respects the first constraint of 1 km resolution. It should be noted that this resolution is better than the 1 km expected with a 80 m radius configuration. Regarding the sidelobes level constraint it can be observed that the configuration fails with some lobes exceeding -10 dB in the glistening zone. It should be mentioned that the low number of samples (29) and the not so homogeneous density (still more elements at the center of the system) degrades the continuous aperture approximation.



(a) 29 antenna layout

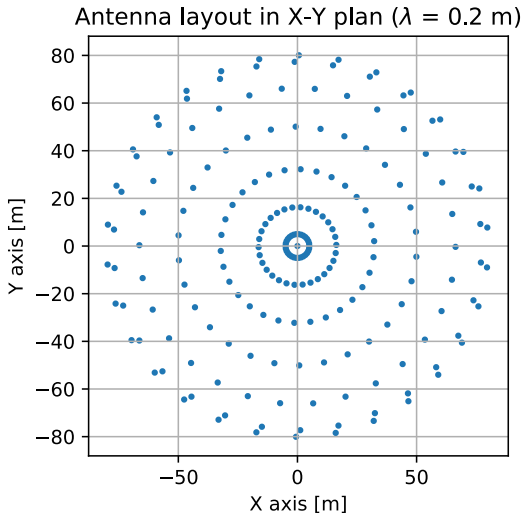


(b) Radiation pattern

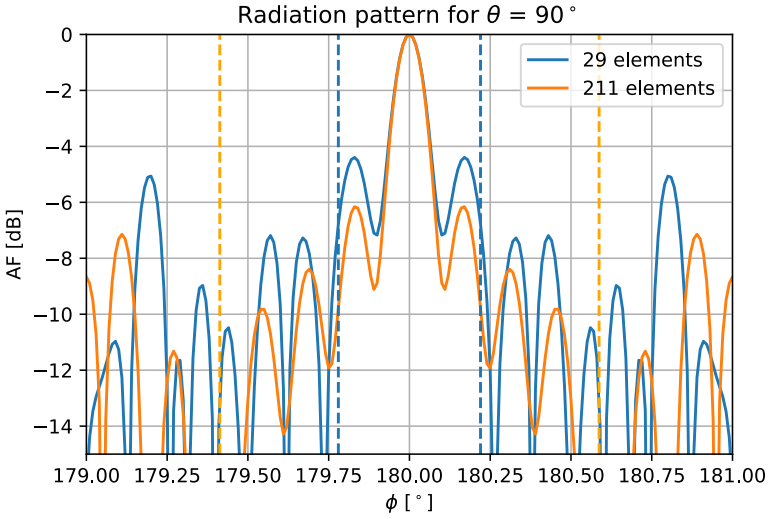
Figure 8: Resolution = 0.79 km

Therefore Figure 9 proposes to analyze the results that could be expected if working with more samples, assuming that way enhancing the continuous aperture approximation.

Figure 9(a) shows the layout of a 30 arms × 7 elements (to which the master CubeSat is added). It should be noted that the smallest distance between two



(a) 211 antenna layout



(b) Array factor comparison

Figure 9: Higher sampled configuration comparison

elements is equal to 1.05 m. This means that this configuration would be feasible regarding minimum distance requirements.

A comparison of the respective array factors of the 29 and 211 element configurations is proposed in Figure 9(b) in a range of 2° around the main lobe. The first

observation is that the 211 element array factor exhibits lower sidelobes, the highest being below -6 dB. This suggests that the -10 dB restriction could be reachable if the number of elements is further increased. It should also be noted that the beamwidth at -3 dB has remained constant. This confirms that the beamwidth is mainly dependent on the radius of the layout.

It is suggested in [16] that the radiation pattern of an approximated continuous aperture can be divided in two regions. The first region, centered around the main beam, should behave like the equivalent continuous aperture. An example of a continuous aperture radiation pattern was given in Figure 5 in Section 3.3.2. The second region would result from an incoherent contribution from antennas leading to higher and unpredictable lobes. Equation (15) is proposed to estimate the limit between the continuous-type radiation zone and the so-called non-coherent region.

$$\frac{2\pi}{k\beta_t} \approx 2d = 2\sqrt{\frac{\pi R^2}{N}} \tag{15}$$

where $\beta_t = \sin\phi_t$ is the limit between both zones and d is the average spacing between elements. The continuous-type region is thus expected to be included in

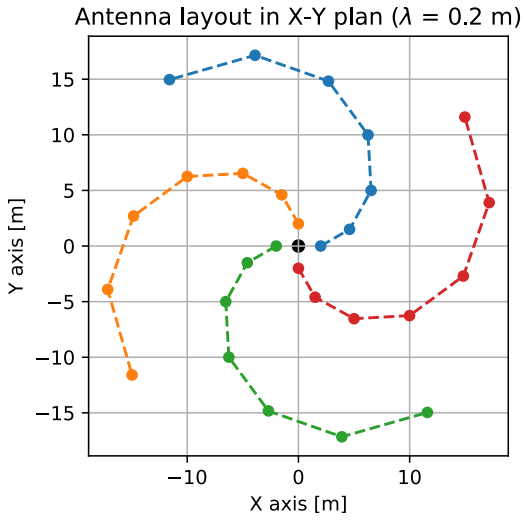
$$180 - \sin^{-1}(\beta_t) \leq \phi \leq 180 + \sin^{-1}(\beta_t) \tag{16}$$

with $\sin^{-1}(\beta_t) \approx 0.22°$ if working with a 80 m radius and 29 elements. Even if this value should only be taken as an order of magnitude, it indicates that this small number of elements necessarily implies a tiny *coherent* region. An increase in the number of elements leads to a larger *coherent* region such that $\sin^{-1}(\beta_t) \approx 0.59°$ if working with 211 elements. Vertical dashed lines in Figure 9(b) represent the estimated limits between the two regions. It can be checked that the theoretical limits seem to fit the actual behavior of the sidelobe and that the 29 element system was a good approximation of the continuous aperture on the first sidelobe.

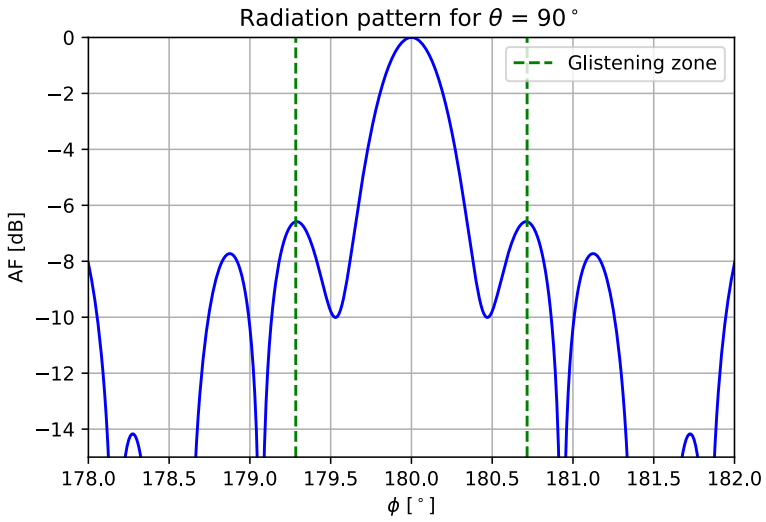
If limiting the sidelobes to -10 dB seems complicated given the low number of elements, it could be interesting to reduce the radius of the configuration and try to exclude secondary lobes of the glistening zone. From this perspective, it is interesting to know which is the smallest beamwidth that is possible to obtain in these conditions. Such a configuration is proposed in Figure 10.

Figure 10(a) shows the 4 arms $\times$ 7 elements layout. The radius is equal to 18 m. Even if the system is smaller, elements are at least 2 m apart.

The radiation pattern of this configuration is shown in Figure 10(b). As expected, lobes are larger and it was possible to get rid of the sidelobes from the glistening



(a) 29 antenna layout



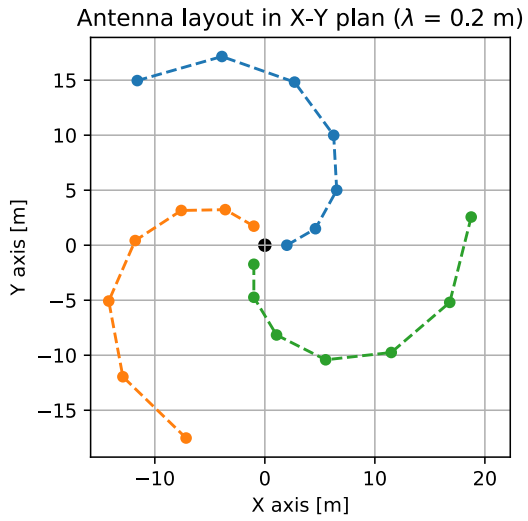
(b) Radiation pattern

Figure 10: Resolution = 3.3 km

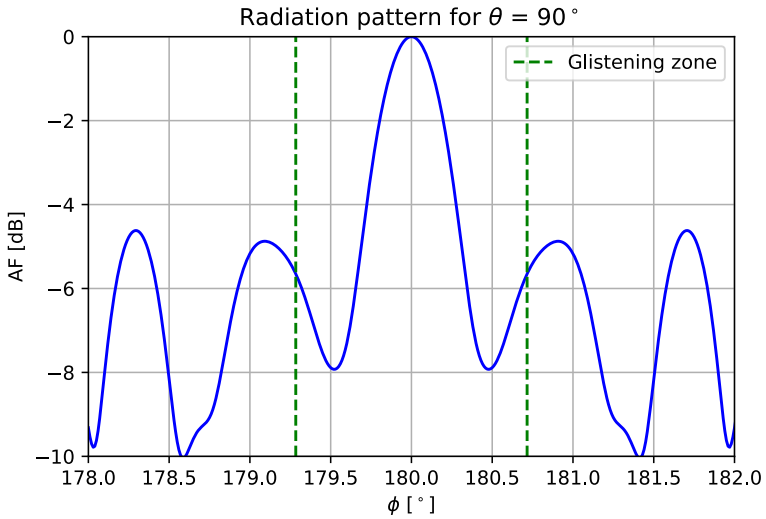
zone. Indeed equation (11) revealed that the beamwidth was inversely proportional to the radius of the equivalent aperture. The obtained resolution is equal to 3.3 km which is larger than the 1 km constraint.

As the equivalent continuous aperture is smaller than with a 80 m radius while the

number of samples has remained constant, it can be assumed that the approximation is better. Therefore it might be decided to reduce the number of elements (for cost reasons for instance). Results of a 3 arms $\times$ 7 elements layout is proposed in Figure 11 with the layout shown in Figure 11(a) and the radiation pattern shown in Figure 11(b).



(a) 22 antenna layout



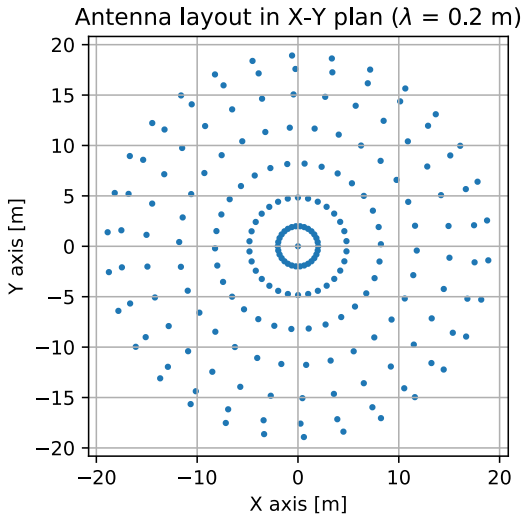
(b) Radiation pattern

Figure 11: Resolution = 3.3 km

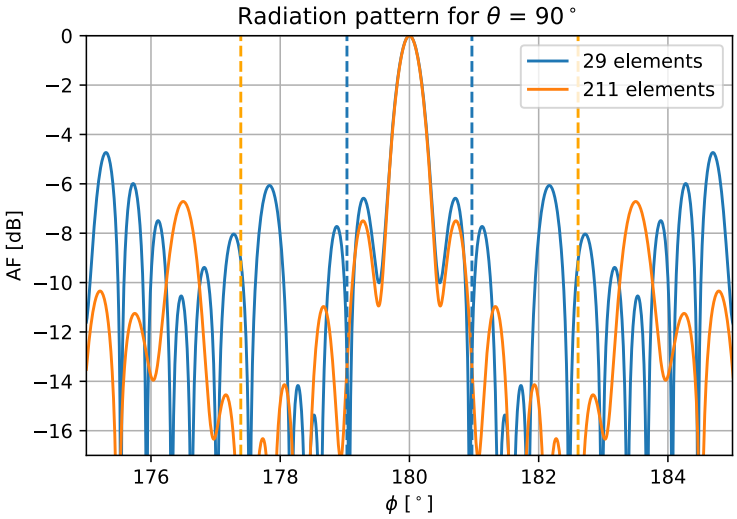
It can be observed that the resulting increase in the sidelobes level does not affect the array factor in the glistening zone as they were excluded from it. However, the reduction of the number of elements will probably cause a reduction in the maximum directivity of the system.

Working with a smaller system also allows to increase the continuous-type radiation zone (see equation (15)). With this 18 m radius configuration the *coherent* region limit is located at $\sin^{-1}(\beta_t) \approx 0.967°$. This means that this region is larger than the glistening zone allowing to have a continuous-type behavior in the entire glistening zone. A comparison with a significantly higher sampled aperture is again shown in Figure 12. The 30 arms $\times$ 7 elements layout is given in Figure 12(a). In this case the smallest distance between two elements is equal to 0.42 m which does no longer respect the minimal acceptable distance of 1 m. However, this configuration is only studied in a theoretical way. A comparison between radiation patterns of 29 and 211 element configurations is shown in Figure 12(b) for a 10° range around the main beam. Limits between *coherent* and *non-coherent* regions are represented with the vertical dashed lines.

As for the 80 m radius configuration, the pattern in the *coherent* zone is a good approximation of the higher sampled configuration.



(a) 211 antenna layout



(b) Array factor comparison

Figure 12: Higher sampled configuration comparison

4 Localization

4.1 Motivation

As explained in Section 3, the scattering of the GNSS signal on the Earth's surface results in the presence of a glistening zone around the specular point. This results in the reflection of multiple signals received by the GNSS-R receiver (the array composed of the CubeSats in formation). An unscaled drawing is shown in Figure 13. The separation and angle-of-arrival estimation of these reflected signals using

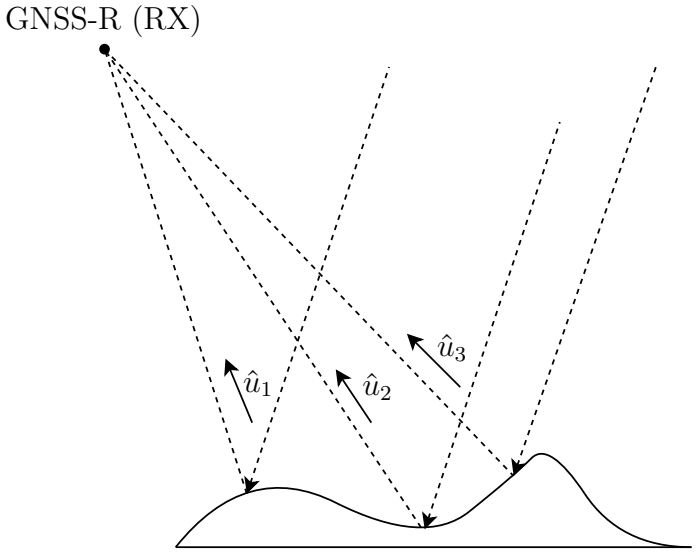


Figure 13: Glistening zone source localization

digital beamforming allows the creation of an image of the observed region. As a first step, this section attempts to demonstrate that a small array of two CubeSats is able to perform single-source localization using the MUSIC beamforming algorithm and to solve CubeSat synchronization constraints.

4.2 CubeSats synchronization

When performing a beamforming operation, an intermediate frequency (IF) demodulation chain is used in general. A schematic of the circuit for one receiver antenna is given in Figure 14. It is mainly composed of a filtering and amplification chain of the received signal (RF domain), a power splitter (PS) to divide the signal in two branches, a local oscillator (LO) splitted and phase shifted by 90° in one of them, two mixers and a filtering and amplification chain in each branch (IF domain). As an array is composed of multiple antennas, the single local oscillator needs to be transmitted to each element.

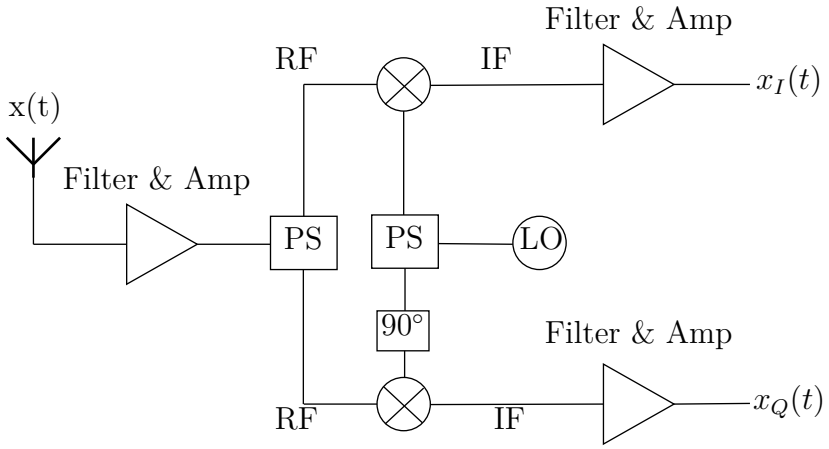


Figure 14: Heterodyne receiver

The aim of such a receiver is to downconvert the RF signal to a predetermined intermediate frequency (IF) so that

$$f_{IF} = f_{RF} - f_{LO} \tag{17}$$

where the received signal $x(t)$ is multiplied by the signal generated by the local oscillator. f_{RF} refers to the frequency of the RF received signal and f_{LO} to the frequency of the local oscillator. As this multiplication process generates a signal at double the frequency, it needs to be filtered again in the IF domain. In-phase and in-quadrature components are obtained using equation (18) where LPF refers to the Low-Pass filtering process.

$$\begin{cases} x_I(t) &= LPF\left\{x(t)\cos(2\pi f_{LO}t)\right\} \\ x_Q(t) &= LPF\left\{x(t)\sin(2\pi f_{LO}t)\right\} \end{cases} \tag{18}$$

However, in the context of an antenna array where each element is on a CubeSat, the transmission of the local oscillator is not trivial anymore as no wired transmission is possible [18], [19], [20].

The objective is thus to develop a solution that enables to perform regular beamforming taking into account the following constraints :

- Wireless transmission of the LO
- Element spacing variation

The first constraint is the single viable synchronization system. It is indeed not feasible to imagine each CubeSat having its own oscillator as they would not be

able to beat exactly at the same frequency. The second constraint appears to be the main issue. For physical reasons, the antenna array will not remain in a fixed configuration when orbiting around the Earth [21]. This means that the phase of the local oscillator signal of each CubeSat can change over time. Indeed, when the distance between a slave and the master CubeSat changes, a phase shift $\Delta\phi$ proportional to the propagation time difference is induced :

$$\Delta\phi = \omega\Delta\tau \tag{19}$$

with ω the pulsation of the signal and $\Delta\tau$ the time delay.

4.3 Solution

The general principle consists in a master CubeSat containing the local oscillator and responsible for its transmission to the other ones called the slave CubeSats.

A simplified schematic of a (slave) CubeSat prototype is available in Figure 15.

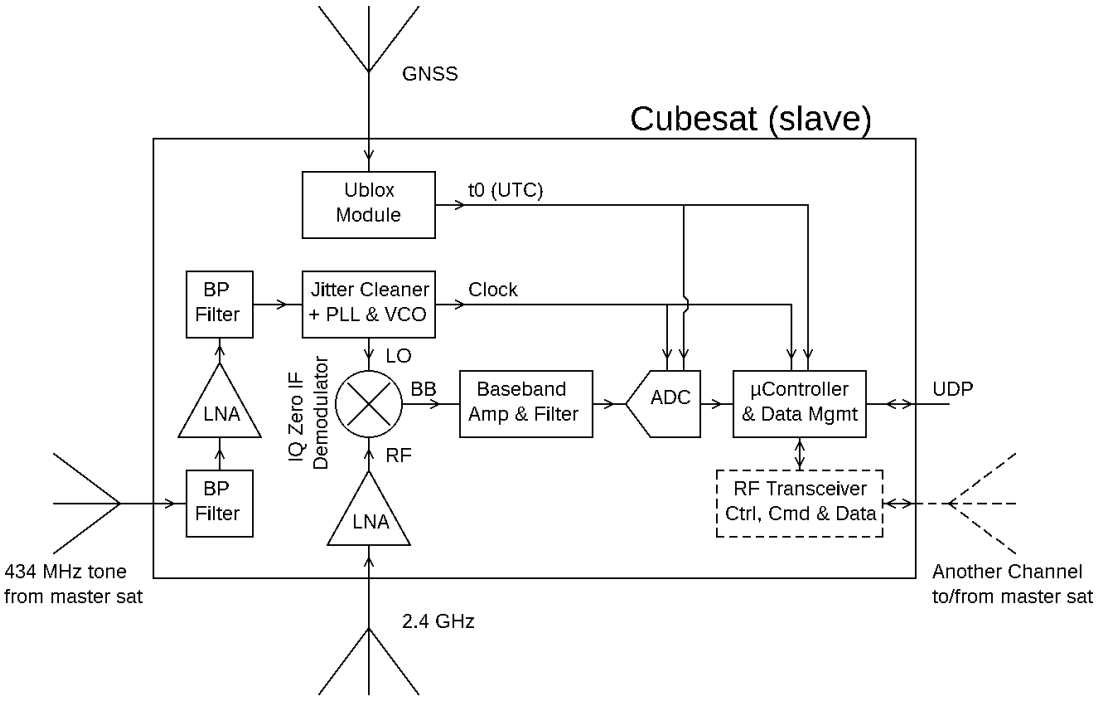


Figure 15: CubeSat prototype [1]

The CubeSat has three receiver antennas to get :

[1]This design is the work of Maxime Drouguet

- The direct GNSS signal
- The reflected GNSS signal
- A 434 MHz signal

In this framework of prototype characterization, the GNSS-R signal is simulated by an externally generated 2.4 GHz tone. Considering the demodulation it was decided to perform an *IQ Zero IF* one. This means that the local oscillator should be at the same frequency as the carrier signal or very close to it. This is a particular case of the IF demodulation described in the previous section.

In the case of a *Zero IF* demodulator the intermediate frequency is the DC one such that $f_{IF} = 0$ and

$$f_{RF} \approx f_{LO} \tag{20}$$

As the shared signal between the CubeSats is pulsed at 434 MHz, a phase-locked loop (PLL) is required to generate a 2.4 GHz signal with the received 434 MHz tone as its input. Both RF and baseband (IF equivalent) filtering and amplification chain are present as introduced in Figure 14. It should be noted that 2.4 GHz is not a multiple of 434 MHz. To achieve the conversion, a fractional-N PLL is required. N is the multiplication factor such that

$$f_{\text{out}} = N \cdot f_{\text{in}} \tag{21}$$

where f_{in} and f_{out} are the input and output frequencies of the PLL. In a fractional-N PLL an internal circuitry allows the factor N to dynamically vary between N and $N + 1$ in an appropriate proportion so that an average division ratio can be achieved [22].

A microcontroller manages the synchronization between the direct GNSS signal and the reflected signal. Finally, a UDP connection allows the data transfer for software signal processing (e.g. beamforming).

4.4 Proof of concept

Given the solution presented in Section 4.2, the rest of this paper will attempt to demonstrate that the prototype is functional by performing radar localization (more precisely, estimation of the direction of arrival) using an antenna array composed of two CubeSat prototypes. Concretely, one would like to demonstrate that the system is able to

- track a source signal when elements are fixed
- track a fixed source signal when elements spacing is varying

4.5 MUSIC

4.5.1 Motivation

As explained in Section 4.4, the system should be able to locate the source signal (dynamically). This will be the reflected GNSS signal in the final application, whereas in this experimental setup, it is a generated 2.4 GHz signal. As stated previously, a phase shift (depending on the angle of arrival of the source) is induced by the spacing between two antennas. This can be formalized by a steering vector representing the phase shift of each receiving antenna with respect to the reference one. This vector is given in equation (22) for an array composed of N antennas.

$$\boldsymbol{\alpha}(\theta) = \left[1, e^{j2\pi \frac{d \sin\theta}{\lambda}}, e^{j2\pi \frac{2d \sin\theta}{\lambda}}, ..., e^{j2\pi \frac{(N-1)d \sin\theta}{\lambda}}\right]^H \tag{22}$$

The situation is illustrated in Figure 16, where antennas placed linearly with constant spacing d are indexed from 0 to N-1.

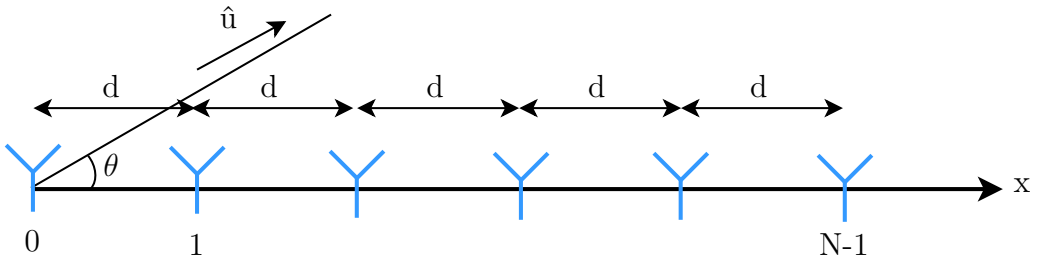


Figure 16: Linear array with constant spacing

Consider an Additive White Gaussian Noise (AWGN) channel with the following model for a single source:

$$\mathbf{x} = \boldsymbol{\alpha}(\theta)\mathbf{s} + \sigma^2\mathbf{I} \tag{23}$$

with $\mathbf{x} \in \mathbb{C}^{1\times N}$ the array output, $\mathbf{s} \in \mathbb{C}^{N\times 1}$ the incoming signal and $\sigma^2\mathbf{I}$ the additive white Gaussian noise covariance.

In order to estimate the direction of arrival, several techniques, described in [23] and [24], are known. These include the calculation of a spatial spectrum:

- Matched filtering : the steering vector is correlated with the vector $\mathbf{x}$ such that

$$P(\theta_i) = |\boldsymbol{\alpha}(\theta_i)^H\mathbf{x}| \tag{24}$$

- Minimum Variance Distorsionless Response (MVDR) which tries to minimize the output energy by resolving a Lagrange optimization problem. This is an adaptive beamforming technique. The spectrum is computed like

$$P(\theta_i) = \frac{1}{|\boldsymbol{\alpha}(\theta_i)^H \mathbf{R}_{xx}^{-1} \boldsymbol{\alpha}(\theta_i)|} \tag{25}$$

  where $\mathbf{R}_{xx}$ is the auto-correlation matrix.

- MUltiple SIgnal Classification (MUSIC) algorithm that exploits the eigen vectors of the auto-correlation matrix.

- Phase Comparison Monopulse (PCMP) that looks for the maximum of the ratio between the weighted-difference pattern and the weighted-sum pattern of the antenna array :

$$R(\theta) = \left| \frac{\mathbf{w}_{diff}(\theta_i)\mathbf{x}}{\mathbf{w}_{sum}(\theta_i)\mathbf{x}} \right| \tag{26}$$

  The computation of the weights involves virtually splitting the array in half, assigning each part with corresponding phase values to generate the sum and difference signal.

- l1 - optimization that tries to solve the following constrained problem

$$\min_s ||\hat{\mathbf{s}}||_1 \text{ s.t. } ||\mathbf{A}(\theta)\mathbf{s} - \sigma^2\mathbf{I}||_2 \leq \epsilon \tag{27}$$

  where each column of $\mathbf{A}$ contains the steering vector $\boldsymbol{\alpha}(\theta_i)$.

They all have the peculiarity of directly exploiting the phase shift induced by the spatial separation between the receiving antennas. Since the MUSIC algorithm has been shown to outperform the others, it has been chosen for the present application [23].

4.5.2 Theoretical description

The Multiple Signal Classification, first described by Schmidt in [25], proposes an unbiased estimate of the directions of arrival of the D multiple signals arriving in an array of M antennas. It starts with the following model :

$$\mathbf{x} = \mathbf{A}\mathbf{s} + \mathbf{w} \tag{28}$$

with $\mathbf{x} \in \mathbb{C}^{M\times 1}$, $\mathbf{A} \in \mathbb{C}^{M\times D}$ the matrix containing the steering vectors, $\mathbf{s} \in \mathbb{C}^{D\times 1}$ is the incident signal vector and $\mathbf{w} \in \mathbb{C}^{M\times 1}$ is the noise vector. By assuming that the number of incident signals D is smaller than the number of array elements M and

that the incident signals and the noise are uncorrelated, the $\mathbf{M} \times \mathbf{M}$ covariance matrix can be obtained as :

$$\mathbf{R_x} = \overline{\mathbf{xx^H}} = \mathbf{A}\overline{\mathbf{ss}}\mathbf{A^H} + \overline{\mathbf{ww^H}} \tag{29}$$

which can be written as

$$\mathbf{R_x} = \mathbf{A}\mathbf{R_s}\mathbf{A^H} + \sigma^2\mathbf{I} \tag{30}$$

$\mathbf{R_x}$ can then be decomposed into a signal subspace $\mathcal{U}_S$ spanned by the eigenvectors associated with the D largest eigenvalues, and a noise subspace $\mathcal{U}_n$ spanned by the eigenvectors associated with the M-D other eigenvalues, such that

$$\mathcal{U}_S \perp \mathcal{U}_n \tag{31}$$

Indeed, $\mathbf{R_x}$ being Hermitian, its eigenvectors are orthogonal.The MUSIC technique then defines a squared norm

$$d^2 = ||\mathbf{U_N^H a}||^2 = \mathbf{a^H U_N U_N^H a} \tag{32}$$

where $\mathbf{a} = [1 \ e^{j\omega} \ ... \ e^{j(M-1)\omega}]^T$ is the candidate steering vector and $\mathbf{U_N}$ is the matrix of eigenvectors associated with the noise subspace. At this stage one can note that this algorithm requires a good a priori guess of the number D of sources.

Finally the direction of arrival is estimated by locating maxima of the MUSIC spectrum

$$\mathbf{P}(\theta) = \frac{1}{\mathbf{a^H U_N U_N^H a}} \tag{33}$$

4.5.3 Algorithm

In the current experimental setup, the direction of arrival of $D = 1$ source signal to an array composed of $M = 2$ elements should be estimated. The system of coordinates is shown in Figure 17.

This means that there are four signals at the output of the array: I_0, Q_0, I_1, Q_1, the in-phase and quadrature signals of each element. The complex signal is recomposed as follows

$$\mathbf{X} = \begin{bmatrix} \mathbf{x_0} \\ \mathbf{x_1} \end{bmatrix} = \begin{bmatrix} \mathbf{I_0 + jQ_0} \\ \mathbf{I_1 + jQ_1} \end{bmatrix} \tag{34}$$

where $\mathbf{X}$ is a $2 \times N$ matrix, N being the number of samples. An estimate of the auto-correlation matrix $\mathbf{R_x}$ is obtained by performing the sample correlation :

$$\widehat{\mathbf{R_x}} = \frac{1}{N}\mathbf{X}\mathbf{X^H} \tag{35}$$

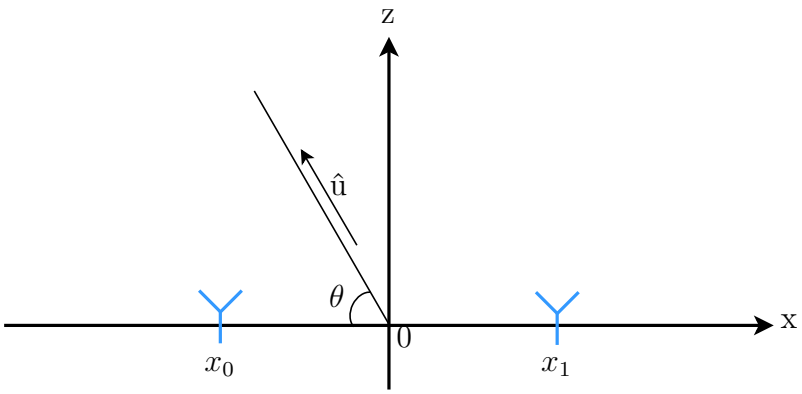


Figure 17: System of coordinates

where $\widehat{\mathbf{R_x}} \in \mathbb{C}^{2\times 2}$. Each coefficient $r_{ij} = \widehat{\mathbf{R_x}}(i,j)$ represents the correlation between the respective signals of elements i and j, $r_{11} = r_{22}$ expecting to be purely real and $r_{12} = \bar{r_{21}}$ containing the phase difference. Indeed, as demonstrated in [26] and recalled in [27] the phase shift between two signals u_1, u_2 can be directly obtained from the correlation coefficients

$$\phi = \cos^{-1}\left(\frac{R_{u_1 u_2}}{\sqrt{R_{u_1} R_{u_2}}}\right) \tag{36}$$

However this information is not directly used by the MUSIC algorithm.

An eigen decomposition is then performed on $\widehat{\mathbf{R_x}}$ giving two eigenvalues λ_0, λ_1 and their associated eigenvectors $\mathbf{v_0}, \mathbf{v_1}$. The noise space does correspond to the vector associated to the lowest eigenvalue and is denoted $\mathbf{U_N} \in \mathbb{C}^{2\times 1}$.

Finally, for each candidate direction, a steering vector

$$\mathbf{a}(\theta_i) = \begin{bmatrix} e^{jkx_0\cos\theta_i} & e^{jkx_1\cos\theta_i} \end{bmatrix} \tag{37}$$

is used to compute the associated MUSIC power

$$P_{MUSIC}(\theta_i)[dB] = 10\log\left(\left|\frac{1}{\mathbf{a^H U_N U_N^H a}}\right|\right) \tag{38}$$

The direction of arrival (DOA) is estimated by searching the θ_i angle that maximizes $\mathrm{P_{MUSIC}}$.

4.5.4 Ambiguity

It should be noted that if the distance between both antennas exceeds $\lambda/2$, multiple peaks will appear for a single source. Indeed, the MUSIC spectrum is nothing else than a digital beamforming technique that is directly related to the classical *delay and sum* beamforming. When working with $D = 1$ source signal a direct relationship between the MUSIC power and the classical beamforming can be obtained. Indeed, as described in [28], if starting again from equation (33) and recalling that

$$\mathbf{I} = \mathbf{U_S}\mathbf{U_S^H} + \mathbf{U_N^H}\mathbf{U_N} \tag{39}$$

the MUSIC power can be written as

$$\mathbf{P}(\theta_i) = \frac{1}{\mathbf{a^H}(\mathbf{I} - \mathbf{U_S}\mathbf{U_S^H})\mathbf{a}} \tag{40}$$

or

$$\mathbf{P}(\theta_i) = \frac{1}{\mathbf{a^H}\mathbf{I}\mathbf{a} - \mathbf{a^H}\mathbf{U_S}\mathbf{U_S^H}\mathbf{a}} \tag{41}$$

The first term of the denominator is equal to the number of antennas $M = 2$. The second term can be developed in the special case of $D = 1$ source signal. In this case the matrix of eigenvector associated to the signal subspace is equal to the steering vector associated to the source signal direction $\mathbf{a}(\theta_s)$ with θ_s the angle of arrival of the source. One can thus define the squared norm

$$\mathbf{P'}(\theta_i) = \frac{1}{M}||\mathbf{a^H}(\theta_s)\mathbf{a}(\theta_i)||^2 \tag{42}$$

which is nothing else than the classical beamforming response. The relationship between the MUSIC response and the classical *delay and sum* beamforming can thus be written as

$$\mathbf{P}(\theta_i) = \frac{1/M}{1 - \mathbf{P'}(\theta_i)/M} \tag{43}$$

However, the multiplicity of sources will not be considered as an issue. Indeed, as developed in Section 3, in the final application, the antenna array will steer the beam in the direction of a given specular zone, the main beam will be very narrow and the array will be designed such that sidelobes are limited to a certain level in the glistening zone.

4.5.5 Calibration

The technique introduced above assumes that the whole phase shift between the two signals comes from the direction of arrival. However, due to the hardware

components, $\mathbf{x_0}$ and $\mathbf{x_1}$ are out of phase even when the source signal is at $\theta = 90°$ in the system of coordinates described in Figure 17. In a more formal way, for any position of the source signal, the phase shift can be decomposed in two terms :

$$\Delta\phi = \Delta\phi_{\text{const}} + \Delta\phi(\theta) \tag{44}$$

The solution to avoid the effect of this unpredictable phase shift is to perform a calibration. This means storing a correlation matrix at a known position to obtain the fixed phase shift and subtracting this for all further estimations.This is performed through an element-wise division such that

$$X'_{m,n} = \frac{X_{m,n}}{\widehat{R_{\text{cal}}}^{H}_{m,0}} \tag{45}$$

where m is the indice related to the M antennas and n is related to the N samples of the signal.

4.6 Target tracking

The first experiment is to show that it is possible to perform radar localization of a single target source with fixed CubeSats.

4.6.1 Setup

Two types of measurements were performed for the fixed CubeSat configuration:

- The local oscillator is wired from the generator to both CubeSats.
- The local oscillator is transmitted wirelessly to both CubeSats. In this configuration, each CubeSat has two antennas. One for locating the source and one for receiving the 434 MHz local oscillator.

The first configuration is shown in Figure 18. Both CubeSats, named RX0 and RX1, are placed along the x-axis at coordinates x_0 and x_1 respectively, not necessarily at the same distance from the origin. The 2472 MHz source is moved along a line at a constant z distance from the x-axis, trying to estimate θ angles from 0 to 90°. The 434 MHz local oscillator (shown in orange in the figure) is wired directly to RX1, which also transmits the oscillator to RX0.

The second configuration is shown in Figure 19. It can be seen that each CubeSat has two antennas. It is important to note that both CubeSats receive the local oscillator with a time delay due to the difference in distance between each receiver and the 434 MHz transmitter. This time delay causes a phase shift between the

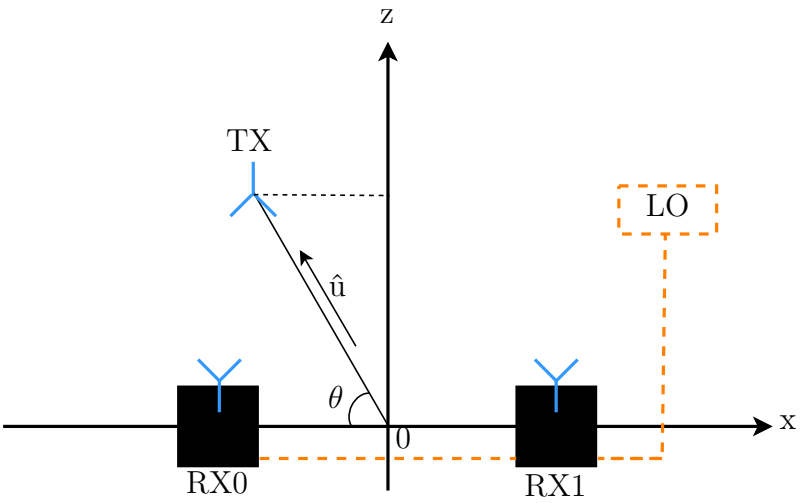


Figure 18: Wired local oscillator link setup

output signals. Indeed, this phase shift is preserved throughout the demodulation chain. It should therefore be taken into account when trying to estimate the direction of arrival. In a configuration where RX0, RX1 and the LO transmitter are fixed (which is the case here), this phase shift remains constant. This problem can be solved simply by calibration, recalling equation (44) and assuming that the constant phase shift is the sum of the hardware and LO phase shifts such that

$$\Delta\phi_{\text{const}} = \Delta\phi_{\text{hard}} + \Delta\phi_{\text{LO}} \tag{46}$$

Figure 20 shows both CubeSat prototypes. One of them is placed on a mobile arm remotely controlled allowing to move the CubeSat with a 1 mm accuracy. The CubeSats are powered by the voltage source visible on the right and a controller (visible in the middle of the CubeSats) connected to the prototypes simulates the reception of direct GNSS signals. Indeed, those signals can not be received inside the anechoic chamber.

A zoom on a prototype is shown in Figure 21. Two antennas are visible. The first one is a monopole antenna (the vertical red wire) that is used to receive the 434 MHz local oscillator. The second one is a patch antenna for receiving the 2.472 GHz signal. On the upper stage, the amplification chain, connected to the 434 MHz antenna can be seen. A balun (on the right) makes the connection with the PLL (blue board on the second stage, starting from the top). The 2.4 GHz patch antenna is connected to the Low Noise Amplifier followed by the *IQ zero IF demodulator* on the third stage. The microcontroller responsible of the data transfer (through a UDP connection) takes place on the lowest stage.

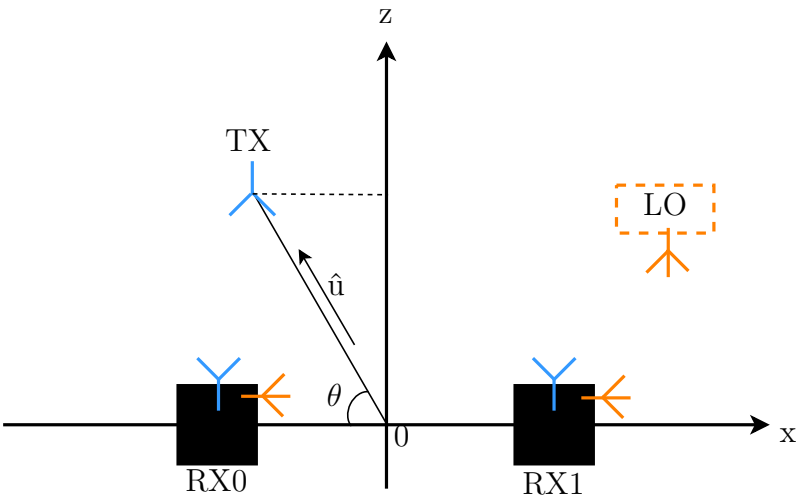


Figure 19: Wireless local oscillator link setup

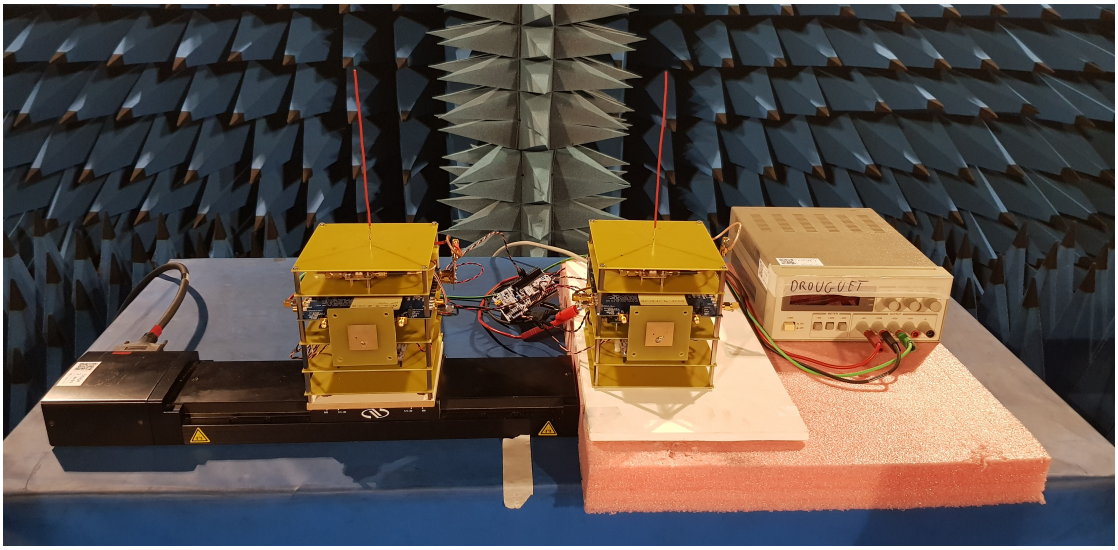


Figure 20: Setup anechoic chamber

Figure 22 shows the small array formed by both CubeSats, the dipole antenna (on the left) for the local oscillator transmission and the patch antenna (on the right) for the 2.472 GHz source signal emission.

4.6.2 Results

Measurements were carried out for three different CubeSat spacings:

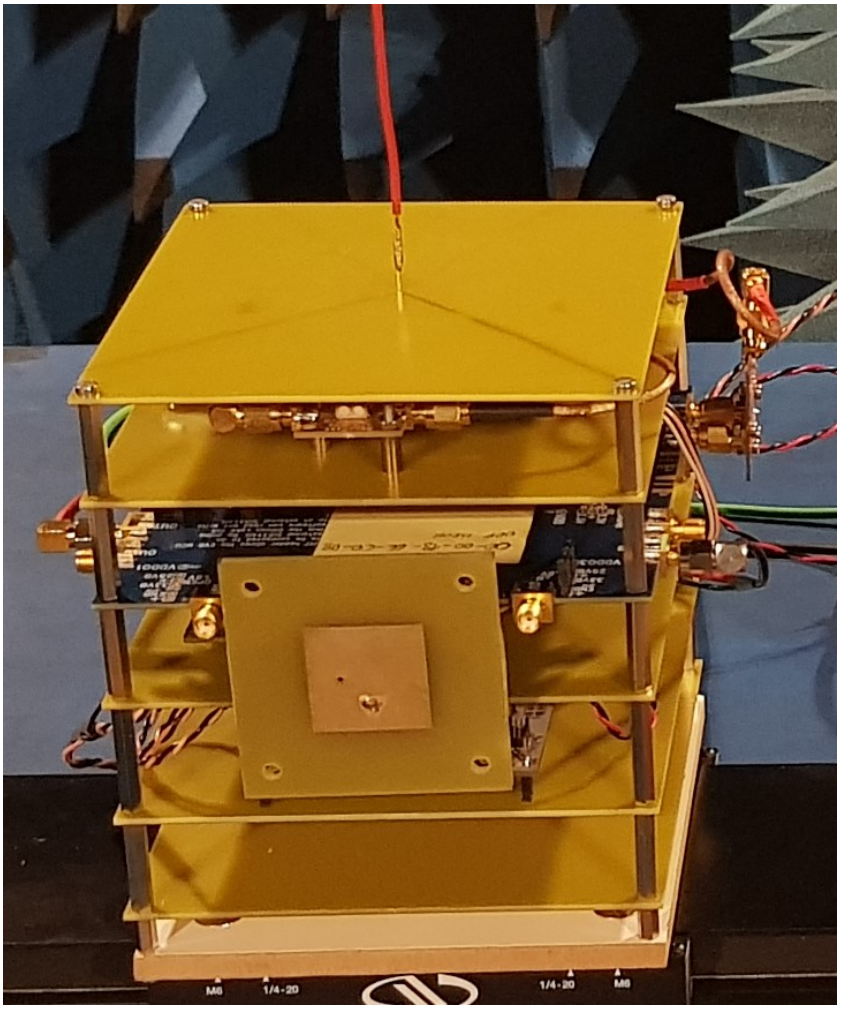


Figure 21: CubeSat prototype

- 18 cm $\approx 1.5\lambda$
- 24 cm $\approx 2\lambda$
- 30 cm $\approx 2.5\lambda$

if working at 2472 MHz which gives $\lambda = 12.1$ cm. Beamformings of the antenna array formed by both CubeSats are given for 1.5λ, 2λ and 2.5λ spacings in Figure 23. Figures 23(a), 23(b) and 23(c) display the beamforming resulting from the MUSIC algorithm. The MUSIC power as defined in equation (38) is expressed in decibels. The horizontal axis spans all the possible directions going from $\theta = 0°$ to $\theta = 180°$ as introduced in Figure 17. The vertical dashed green line indicates the theoretical direction of the source. In all three figures the source was located

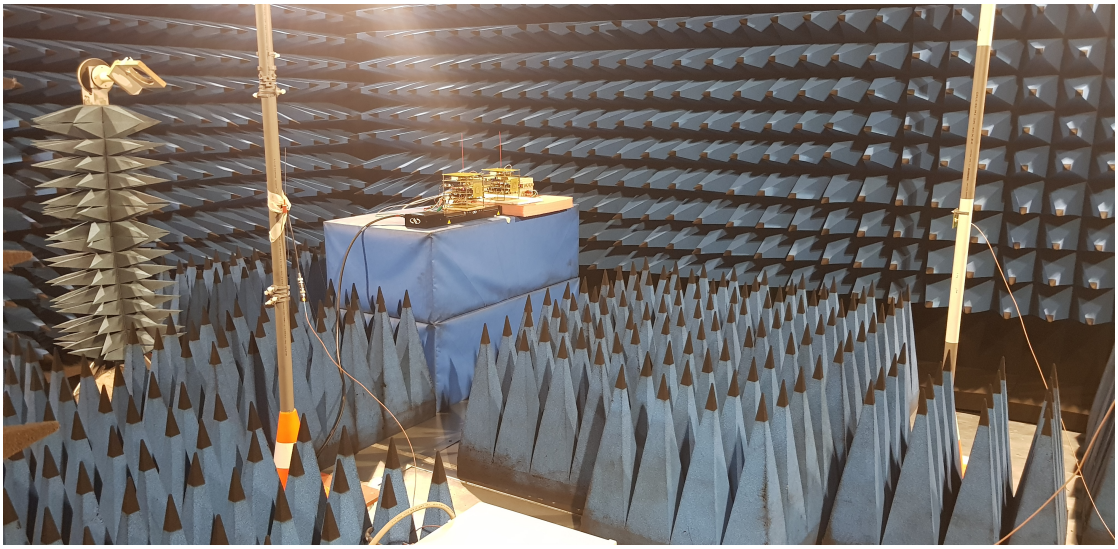
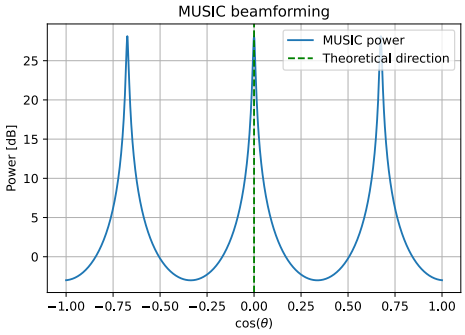


Figure 22: Setup anechoic chamber

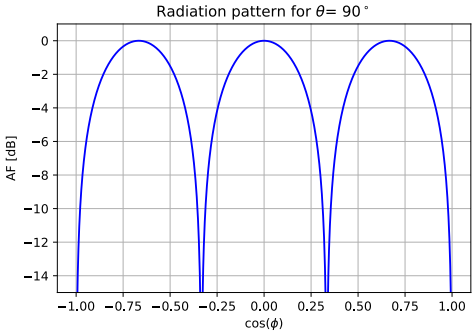
at $\theta = 90°$. As those directions were used to calibrate the system, the theoretical direction and the MUSIC peak match perfectly.

The presence of three peaks comes from the CubeSat spacing being larger than $\lambda/2$, leading to the appearance of grating lobes. As a way of confirmation on the number and placement of these peaks, the array factor was computed as defined in equation (4). Results displayed in Figures 23(d), 23(e) and 23(f) confirm the number of peaks and their location as obtained with MUSIC beamforming. It can be noted that MUSIC peaks are significantly sharper than the lobes obtained through classical beamforming. The sharpness of the lobes in MUSIC beamforming can be explained recalling equation (43) from Section 4.5.4 relative to ambiguity issues. It appears that when $\mathbf{P}'(\theta_i)/M$ is equal to 1 (corresponding to a maximum of the radiation pattern), MUSIC power tends towards infinity. Both beamformings contain the same information but represent it in a different way. As developed in the same section, this ambiguity does not represent an issue.

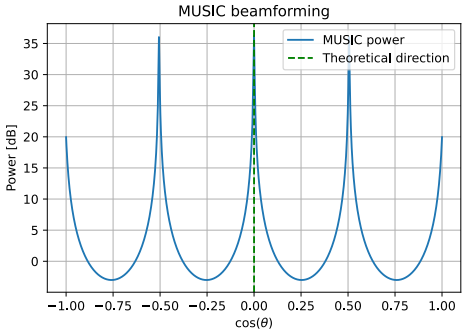
It should be noted that the θ angle introduced in Figure 18 corresponds to the ϕ angle of the system of spherical coordinates (see Figure 2) such that horizontal axis of all Figures in 23 refer to the same angle.



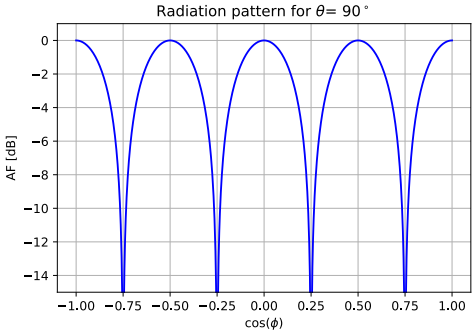
(a) MUSIC - 1.5λ



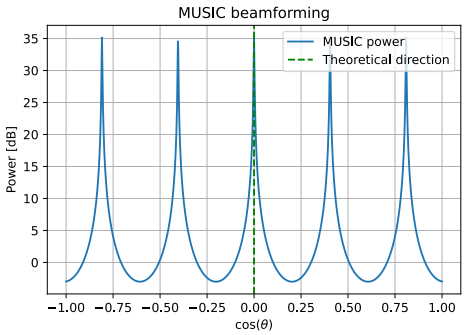
(d) Array factor - 1.5λ



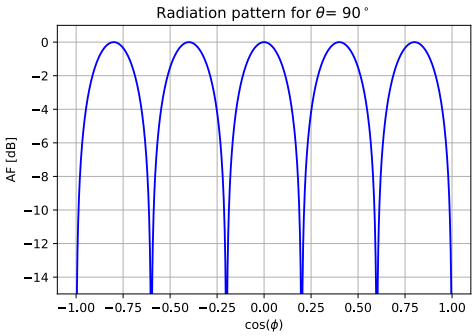
(b) MUSIC - 2λ



(e) Array factor - 2λ



(c) MUSIC - 2.5λ



(f) Array factor - 2.5λ

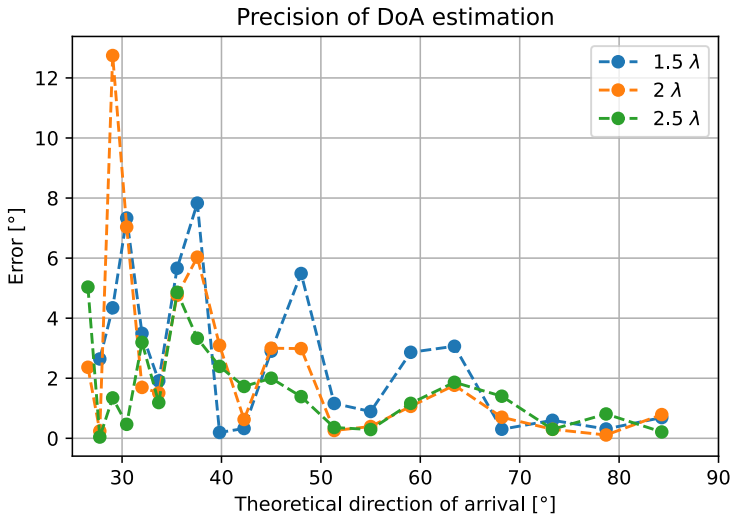
Figure 23: Beamforming for different spacings

The first set of measurements was carried out indoors without any special system to avoid reflections. The source signal was placed 1 m from the center of the array with $\theta = 90°$ for calibration. The results are shown in Figure 24. Figure 24(a) presents the accuracy of the direction of arrival estimation for a set of discrete directions ranging from 25° to 90°. The absolute error in degrees (vertical axis) is given with respect to the theoretical arrival direction (horizontal axis) for the three distances (each in a different color). Some bumps up to 8° (neglecting outliers) can be observed in the precision for the three spacings. As the localization was performed in a small and closed room, it can be assumed that these significant errors come from some reflections and the multiplicity of paths. Figure 24(b) shows a summary of the estimation error. Three box plots (one for each distance) are given with the absolute error in degrees on the vertical axis. Orange horizontal lines give the median error and boxes extend from the first to the third quartile denoted $Q1$ and $Q3$. Whiskers go from $Q1 - 1.5IQR$ to $Q3 + 1.5IQR$, where IQR is the interquartile range defined as $Q3 - Q1$. Outliers are represented with small bullets. It can be seen that the median error goes from 3° with 1.5λ spacing to 1.5° with 2.5λ spacing. It can also be observed that the IQR decreases when the spacing increases. This is expected as the resolution is better for larger spacings. Indeed, a given angle is expressed by a larger phase shift when working with larger element spacings.

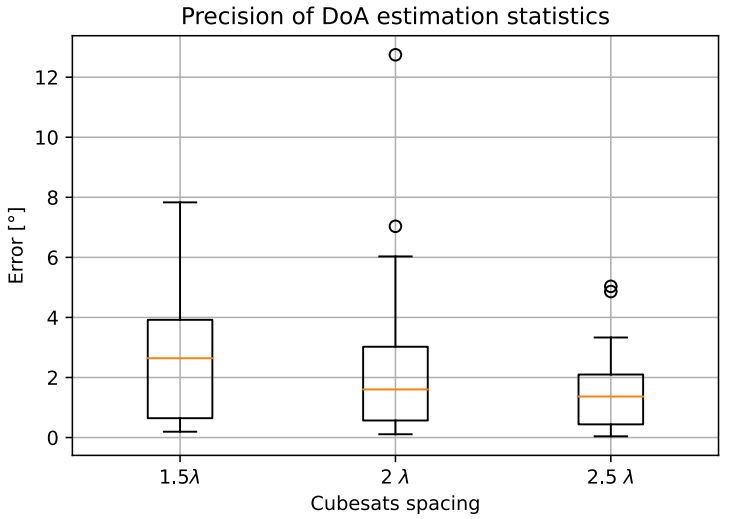
To better characterize the accuracy that can be achieved with the local oscillator wired, the same set of measurements was carried out in an anechoic chamber, which allows most of the reflections to be eliminated. The results are shown in Figure 25. They are again presented with accuracy results in Figure 25(a) and the box plots in Figure 25(b).

Under these conditions, the median error drops below 0.5° at a distance of 1.5λ and to less than 0.2° at a distance of 2.5λ. In the latter, the estimation error never reaches 0.5° (except for one outlier). As in Figure 24, the precision increases with the distance.

Figure 26 provides the results of measurements performed in an anechoic chamber for a direction of arrival estimation with wireless transmission of the local oscillator, as shown in Figure 19. Estimation errors for the three different distances are given in Figure 26(a) and summarized in boxplots in Figure 26(b). Compared to the results obtained in a wired configuration, there is a loss of precision of 0.1 (2.5λ distance) to 0.3° (2λ distance) in terms of median error. A greater variability in precision for the 2λ and 2.5λ distances should also be noted.

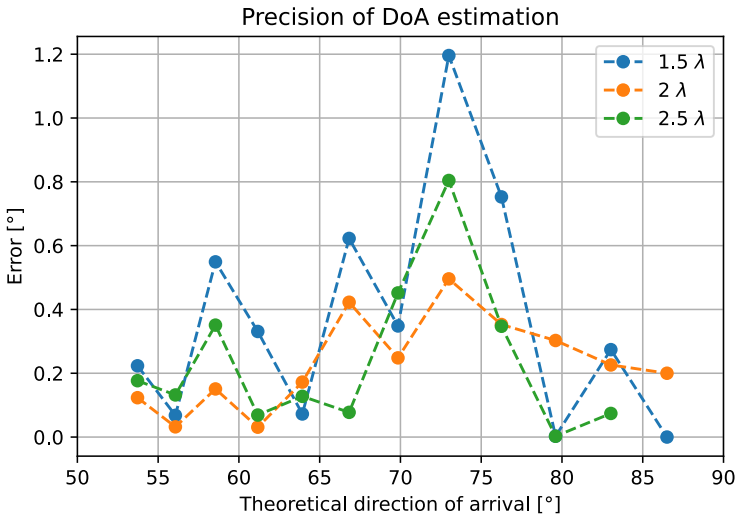


(a) Precision

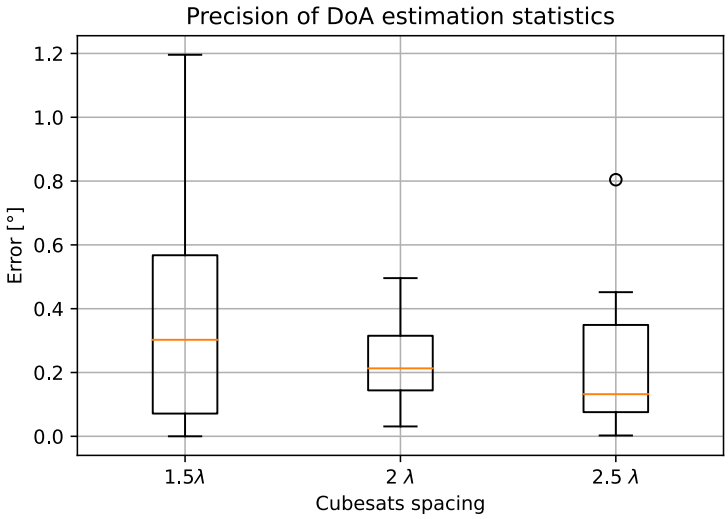


(b) Statistics

Figure 24: Indoor wired local oscillator transmission

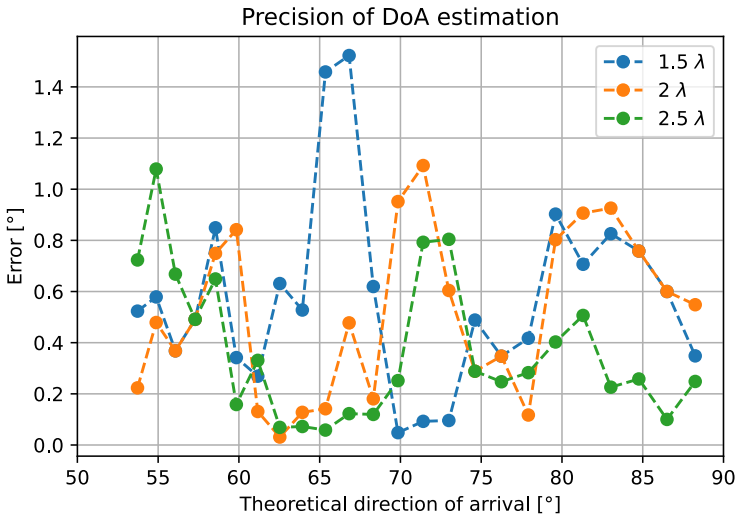


(a) Precision

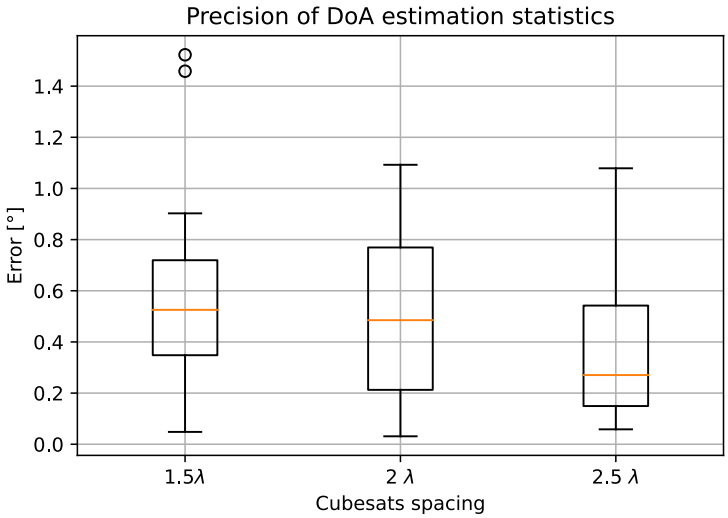


(b) Statistics

Figure 25: Anechoic chamber wired local oscillator transmission



(a) Precision



(b) Statistics

Figure 26: Anechoic chamber wireless local oscillator transmission

4.7 Fixed target

4.7.1 Setup

The second configuration shown in Figure 27 assumes that the 2472 MHz source signal is fixed, for instance in the given direction $\theta = 90°$, while the relative distance between the two CubeSats (and thus the receiving antennas) changes over time. In the current configuration, RX0 remains static while RX1 can move to any $x \geq 0$. Similarly to the configuration with a mobile source signal, a calibration is firstly performed to correct the phase shift induced by the hardware components and the local oscillator distribution by performing localization at a known position. As this known position usually corresponds to $\theta = 90°$, it is evident that keeping the source signal at this same position is not very relevant in practice. However, it allowed to only characterize the influence of the local oscillator transmission while not being disturbed by any other estimation error.

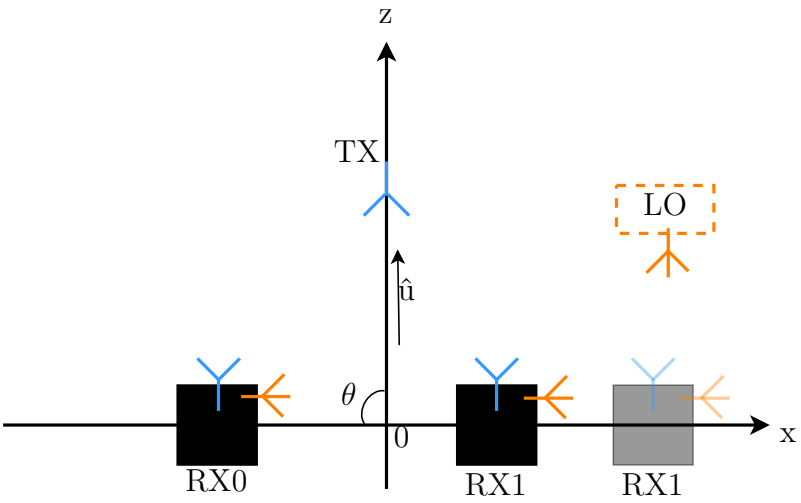


Figure 27: Wireless local oscillator link setup

The aim is to be able to continue to estimate the correct arrival direction without having to re-calibrate. Previously, a new calibration matrix was calculated each time the CubeSat spacing was changed. As explained above, a change in spacing implies a change in arrival delay, resulting in a new angular phase shift on the output signals. Fortunately, if the relative motion of the CubeSats is known (which will always be the case in the final application), it is possible to calculate this phase shift. The situation is illustrated in Figure 28.

The initial situation is represented by the triangle formed by the antenna broadcasting the local oscillator LO (the orange dot) and the two CubeSats RX0 and

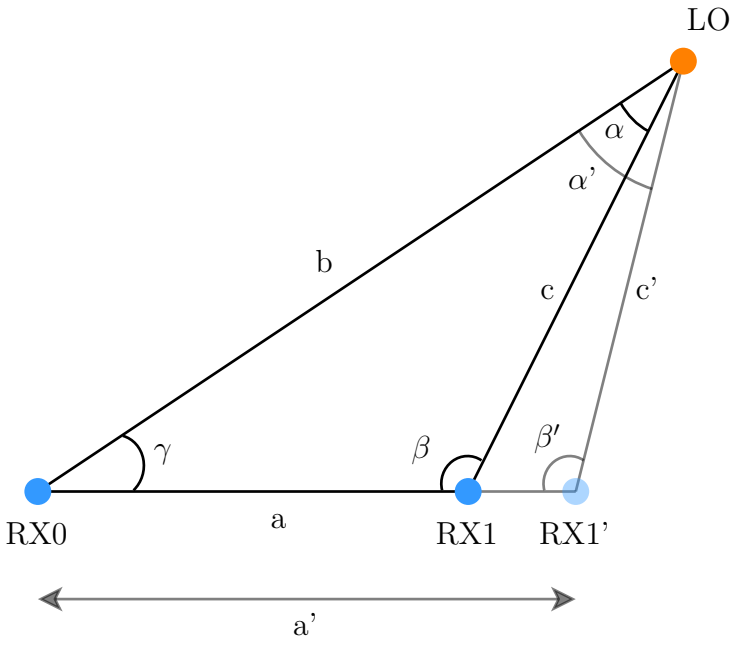


Figure 28: Phase shift correction

RX1 (the blue dots). The three distances a, b and c are assumed to be known. As the triangle is not right, the three angles α, β and γ have to be calculated according to the Al-Kashi theorem, also called *law of cosines*. They are recalled in equations (47) and (48) [29].

$$\begin{cases} c^2 &= a^2 + b^2 - 2ab\cos\gamma \\ a^2 &= b^2 + c^2 - 2bc\cos\alpha \\ b^2 &= a^2 + c^2 - 2ac\cos\beta \end{cases} \tag{47}$$

$$\begin{cases} \gamma = \arccos\left(\frac{a^2+b^2-c^2}{2ab}\right) \\ \alpha = \arccos\left(\frac{b^2+c^2-a^2}{2bc}\right) \\ \beta = \arccos\left(\frac{a^2+c^2-c^2}{2ac}\right) \end{cases} \tag{48}$$

After RX1 moves, a new triangle is formed by the local oscillator LO, the RX0 CubeSat and RX1 at its new position (RX1' in Figure 28). The lengths b and c and the angle γ remain constant. To obtain the phase shift, the new distance between the local oscillator and RX1 must be calculated, i.e. the side c' of the

triangle. It can be obtained by knowing equation (47):

$$c' = \sqrt{a'^2 + b^2 - 2a'b\cos\gamma} \tag{49}$$

where a' is the new distance between RX0 and RX1 and is supposed to be known.

By multiplying the difference of distance by the wavenumber, the angular phase shift can be calculated:

$$\Delta\phi \text{ [rad]} = k\Delta c = k(c' - c) \tag{50}$$

with $k = 2\pi/\lambda$ and $\lambda = 3e8/434e6$.

It should be noted that in the situation shown in Figure 28, Δc is negative and therefore the corrective phase shift $-\Delta\phi$ is positive. Since the propagation path between the local oscillator and RX1 has been shortened, it is required to delay $\mathbf{x_1}$.

In addition to this phase shift $\Delta\phi$, it should be noted that the PLL converts a 434 MHz signal to a 2472 MHz signal and that the phase shift depends on the working frequency, so that :

$$\begin{cases} \Delta\phi_l &= \omega_l\Delta\tau \\ \Delta\phi_h &= \omega_h\Delta\tau \end{cases} \tag{51}$$

where $\Delta\tau$ is the delay caused by the movement of the LO at RX1 and ω_l and ω_h are the low (434 MHz) and high (2472 MHz) pulsations respectively. By isolating $\Delta\tau$ the induced phase shift on the high frequency signal, is obtained as :

$$\Delta\phi_h = \frac{\omega_l}{\omega_h}\Delta\phi_l \tag{52}$$

where the pulsation ratio is 5.7 when operating at 434 MHz and 2472 MHz.

The phase shift is finally corrected by multiplying the complex output signal $\mathbf{x_1}$ as defined in equation (34) by a complex exponential which takes into account the angular phase shift calculated on the low frequency signal (equation (50)) and the correction factor:

$$\mathbf{x_1}^{\text{corrected}} = \mathbf{x_1}e^{-j\Delta\phi_h} = \mathbf{x_1}e^{-j\frac{\omega_h}{\omega_l}\Delta\phi_l} \tag{53}$$

4.7.2 Results

Figure 29 reports the results of a direction of arrival estimation measurement with a wireless local oscillator transmission, a fixed source signal and a varying inter-CubeSat spacing. The error in degrees for multiple distances expressed in multiples of the wavelength λ is given in Figure 29(a). The vertical green dashed line illustrates the distance used for calibration, which is 2.5λ. Therefore the error at this point is equal to zero. Figure 29(b) summarizes the results in a box plot. Looking at the median error, there is a loss of precision of 1.5 to 2° compared to the results shown in Figure 26 (wireless LO transmission with fixed CubeSats). However, this statistic does not appear to be very representative. In fact, the details provided in Figure 29(a) show that the error increases linearly with distance, especially for spacings greater than 2.5λ, reaching more than 8° errors. This means that the system clearly fails to perform localisation under these conditions.

The error, which is linearly dependent on the distance, suggests that some factor other than the 5.7 one (equation (52)) is missing from the phase correction. Further tests showed that the problem was in fact caused by the phase locked loop (PLL). The PLL induces a phase shift strongly dependent on the amplitude of the incoming signal (in this case the 434MHz LO). Since the power of the transmitted LO remains constant, an increase or decrease in the distance between the LO transmitter and the moving CubeSat causes an amplitude variation of the received signal due to free path losses.

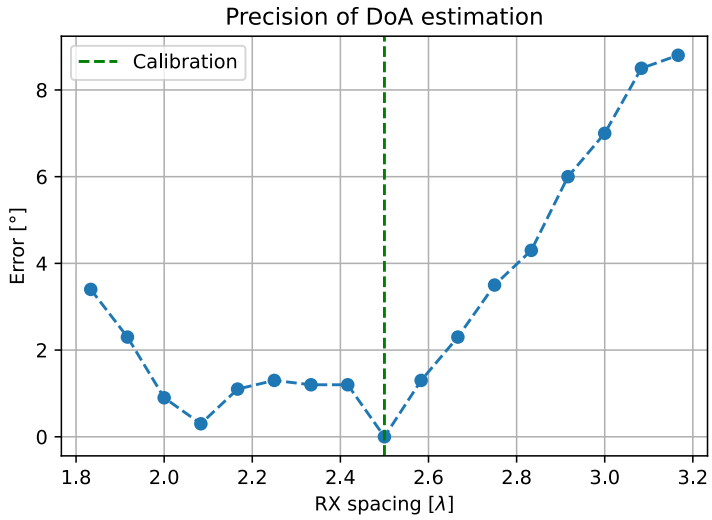
This amplitude-dependent phase shift prevents the system from correctly estimating the direction of arrival. Therefore, this problem must be solved either by being able to predict the amplitude variation or by eliminating it by designing an amplifier chain to saturate the signal.

4.8 Interpretation

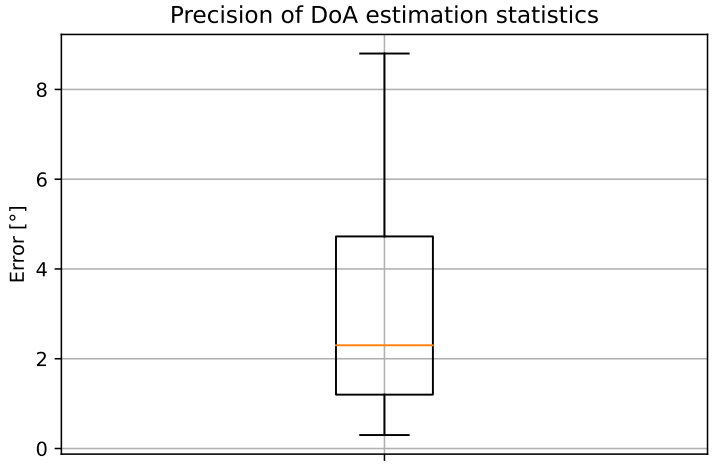
The characterization of the radar system was realized by performing tests in three different conditions :

- Wired LO transmission with fixed CubeSats and mobile source signal
- Wireless LO transmission with fixed CubeSats and mobile source signal
- Wireless LO transmission with a mobile CubeSat and a fixed source signal

The average median errors are given in Table 1.



(a) Precision



(b) Statistics

Figure 29: Fixed source and wireless local oscillator transmission

The wired configuration can be considered as the reference or the best accuracy that can be expected. As developed in Section 3, it is expected to work with a resolution of 1 km in a 10 km diameter glistening zone. Considering that the array will be 400 km away from the Earth's surface, this means that one has to deal with angles between 0.15° and 1.5°. The orders of magnitude are therefore satisfactory for the

	Wired LO	Wireless LO 1	Wireless LO 2
[°]	0.2	0.4	2.3

Table 1: Average median error

first two configurations for a first prototype. The loss of accuracy of 0.2° when using wireless LO transmission is probably mainly due to a deterioration of the SNR. Indeed, the wireless transmission implies an amplification chain (otherwise bypassed) which adds some noise to the signal. However, as mentioned above, the results obtained in the last configuration are not acceptable. Further development is needed to introduce an amplifier chain.

At this point, it is important to mention that the error values discussed should not be considered as representative of what will be possible in the final application. In fact, this experimental setup was realized under specific conditions, in particular with a very good signal-to-noise ratio, and the MUSIC performances are directly proportional to this SNR. This characterization should be considered as a reference for further development. Hardware components, operating frequencies, PLL configuration and signal processing are likely to evolve over the next stages of development. Each modification of the prototype will be accompanied by a new characterization that can be compared with the measurements presented in this thesis.

4.9 PLL

In Section 4.7, it was shown that the PLL is the source of large estimation errors in a mobile CubeSat configuration due to a strongly amplitude-dependent induced phase shift. From this perspective, several solutions are considered:

- Saturating the incoming signal by means of an amplification chain
- Working with an automatic gain control
- Using another PLL

The first solution would theoretically ensure that there is always a signal of the same amplitude at the PLL input. However, this technique has several drawbacks. The main one is that the flat part of the amplifier response is not completely flat. In fact, the saturation induces the appearance of harmonics causing a loss of power at the fundamental frequency. This phenomenon can be observed in Figure 30 which presents the output-input response of the amplifier. It can be seen that the response slightly decreases from -10 db to 10 dB while being in the saturated

part of the response. As a result, a small phase shift (up to a few degrees) can be measured between the input and the output of the amplifier (when working in the flat part of the response). This amplitude-dependent phase shift is illustrated in Figure 31. The input amplitude of the amplifier has been swept from 0 dBm to 10 dBm (corresponding to the saturated part of the amplifier) and the phase shift relative to the one at 0 dBm has been reported. This solution could therefore be a short-term one, assuming the design of appropriate filters and fully characterizing the phase shift induced by the amplifier. It may contribute to rapid small-scale proof of concept of the system, before moving large-scale development plans.

The second solution, although it requires more development, is probably more sustainable. It would indeed allow working with constant amplitude while avoiding the problems caused by signal saturation.

Finally, the search for another PLL that does not induce an amplitude-dependent phase shift could also be considered. More recent PLL technology, e.g. based on ring oscillators, might help solving this issue [30].

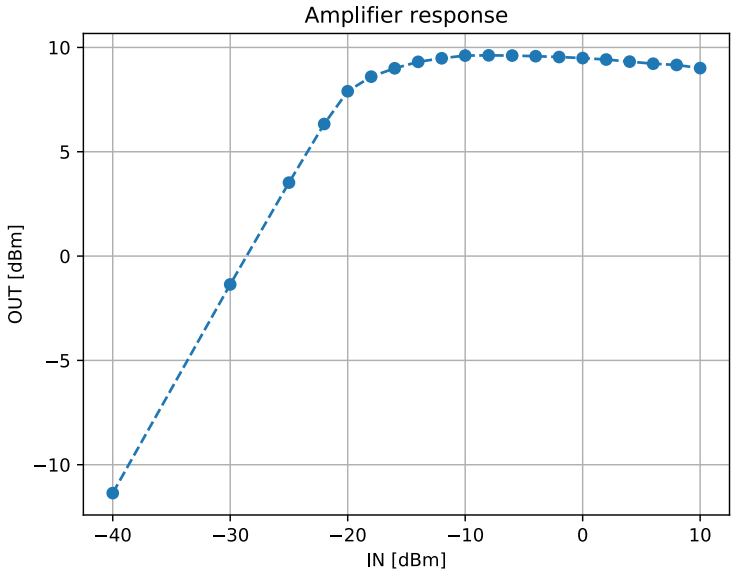


Figure 30: Amplifier response

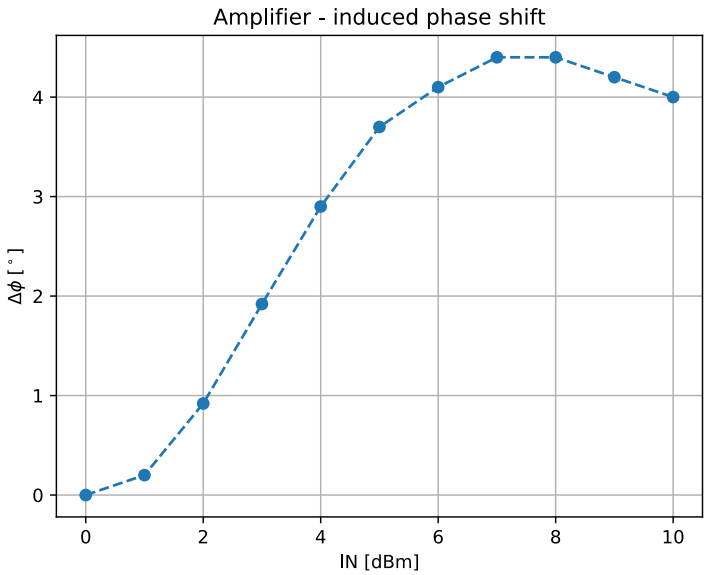


Figure 31: Amplitude-dependent phase shift

5 Conclusion

The current thesis aimed to contribute to the development of an innovative way to perform Earth observation using GNSS reflectometry, that is, the use of CubeSats in formation for antenna array synthesis. It was divided into three main chapters.

Section 2 first introduced the concepts of GNSS and GNSS-R. The wide range of applications, the advantages of GNSS-R over other Earth observation techniques, and the concept of signals of opportunity were described. The use of CubeSats in formation was then motivated. In particular, it was explained how a swarm of CubeSats could address the need for highly directional antennas while drastically reducing mechanical and cost constraints. Finally, the need for digital beamforming was discussed.

Section 3 was related to the search for suitable CubeSat configurations. It started with some quantitative objectives related to resolution (1 km) and sidelobe levels (-10 dB) in a restricted region around the specular point called the *glistening zone*. After introducing the system of spherical coordinates, the methodology was described in detail. The radiation pattern and directivity were defined, the concepts of space tapering and continuous aperture approximation were described, and the use of spirals for configuration elaboration was motivated and detailed. It was shown that it is difficult to achieve resolution and sidelobe level objectives simultaneously and that there is an unavoidable trade-off between resolution and sidelobes. Indeed, it has not been possible to produce a layout that fully satisfies both constraints. However, it has been shown that using spiral configurations, it is possible to achieve continuous-type behavior in a closed region around the main beam, and that the size of this region can be predicted. The direct relationship between the radius of the configuration and the resolution obtained was also demonstrated. Considering the relative predictability of the results, better techniques (than a try-and-fail approach) for local optimization of the element positions could be considered and be part of further work to further reduce the sidelobe levels while keeping constant the number of CubeSats and the equivalent continuous aperture radius.

Section 4 first explained the need to be able to perform digital beamforming and radar localization in the glistening zone for later production of Earth imagery. Synchronization issues related to the inability to transmit the local oscillator through a cable and the proposed solution, including the CubeSat prototype design, were then presented. The use of a fractional N PLL was highlighted. A motivation for the use of the MUSIC algorithm for digital beamforming and a theoretical description followed. The system characterization was then presented with three

experiments. The first experiment with a wired local oscillator transmission served as a reference for more complex experiments. The second experiment with fixed CubeSats and a mobile source showed very satisfactory results with respect to the wired transmission case with a 0.2° degradation on the estimation accuracy. The last experiment, with a fixed source signal and variable CubeSat spacing, failed to perform proper localization. In fact, an amplitude-dependent phase shift induced by the PLL was identified as the cause of the inaccurate estimation. Several solutions were suggested, such as signal saturation or automatic gain control amplifier chains. The use of a PLL in a different technology was also considered.

In conclusion, this thesis is intended to serve as an intermediate step in the demonstration of the core functionality of the interferometer, i.e. synchronization. We hope it can serve as a milestone for further research related to microwave observation of the Earth making use of swarms of CubeSats.

References

[1] *What is Earth Observation?* May 2023. URL: https://www.euspa.europa.eu/european-space/eu-space-programme/what-earth-observation. (accessed: 01.06.2023).

[2] *Putting Earth observation into 'the market perspective'*. Oct. 2021. URL: https://www.esa.int/Applications/Observing_the_Earth/Putting_Earth_observation_into_the_market_perspective. (accessed: 01.06.2023).

[3] Valery U. Zavorotny et al. "Tutorial on Remote Sensing Using GNSS Bistatic Radar of Opportunity". In: *IEEE Geoscience and Remote Sensing Magazine* 2.4 (2014), pp. 8–45. DOI: 10.1109/MGRS.2014.2374220.

[4] *What is GNSS?* Dec. 2021. URL: https://www.euspa.europa.eu/european-space/eu-space-programme/what-gnss. (accessed: 11.05.2023).

[5] Wikipedia contributors. *GNSS reflectometry — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=GNSS_reflectometry&oldid=1147749813. [Online; accessed 11-May-2023]. 2023.

[6] *Types of orbits*. Mar. 2020. URL: https://www.esa.int/Enabling_Support/Space_Transportation/Types_of_orbits#LEO. (accessed: 11.05.2023).

[7] Wikipedia contributors. *Satellite navigation — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Satellite_navigation&oldid=1153673021. [Online; accessed 11-May-2023]. 2023.

[8] C.R. Paul. *Transmission Lines in Digital and Analog Electronic Systems: Signal Integrity and Crosstalk*. Wiley, 2011. ISBN: 9781118058244. URL: https://books.google.be/books?id=NDa9Vl6WBdMC.

[9] J. Puig-Suari et al. "Development of the standard CubeSat deployer and a CubeSat class PicoSatellite". In: *2001 IEEE Aerospace Conference Proceedings (Cat. No.01TH8542)*. Vol. 1. 2001, 1/347–1/353 vol.1. DOI: 10.1109/AERO.2001.931726.

[10] Adam Wuerl and Melissa Wuerl. "Lessons learned for deploying a microsatellite from the International Space Station". In: *2015 IEEE Aerospace Conference*. 2015, pp. 1–12. DOI: 10.1109/AERO.2015.7119213.

[11] Sining Liu et al. “A Survey on CubeSat Missions and Their Antenna Designs”. In: *Electronics* 11.13 (2022). ISSN: 2079-9292. DOI: 10.3390/electronics11132021. URL: https://www.mdpi.com/2079-9292/11/13/2021.

[12] Ugo-Paul Duraffourd. “CubeSats in formation for Earth observation”. Prom. : Craeye, Christophe ; Fisette, Paul. MA thesis. Ecole polytechnique de Louvain, Université catholique de Louvain, 2022.

[13] Wikipédia. *Coordonnées sphériques — Wikipédia, l’encyclopédie libre*. [Online; accessed 15-May-2023]. 2023. URL: %5Curl%7Bhttp://fr.wikipedia.org/w/index.php?title=Coordonn%5C%C3%5C%A9es_sph%5C%C3%5C%A9riques&oldid=202919069%7D.

[14] “IEEE Standard Definitions of Terms for Antennas”. In: *IEEE No 145-1969* (1969), pp. 1–10. DOI: 10.1109/IEEESTD.1969.7366531.

[15] Sokolow Valentin et al. “Scanning GNSS-R Beams from Cubesats Using Sequentially Rotated Deployable Dipoles”. In: *2021 IEEE International Geoscience and Remote Sensing Symposium IGARSS*. 2021, pp. 8538–8541. DOI: 10.1109/IGARSS47720.2021.9554649.

[16] Thibault Clavier et al. “A Global-Local Synthesis Approach for Large Non-Regular Arrays”. In: *IEEE Transactions on Antennas and Propagation* 62.4 (2014), pp. 1596–1606. DOI: 10.1109/TAP.2013.2284816.

[17] Per-Simon Kildal. *Foundations of Antenna Engineering: A Unified Approach for Line-of-Sight and Multipath*. 2015.

[18] G. Vallant et al. “Analog IQ impairments in Zero-IF radar receivers: Analysis, measurements and digital compensation”. In: *2012 IEEE International Instrumentation and Measurement Technology Conference Proceedings*. 2012, pp. 1703–1707. DOI: 10.1109/I2MTC.2012.6229222.

[19] A.A. Abidi. “Direct-conversion radio transceivers for digital communications”. In: *IEEE Journal of Solid-State Circuits* 30.12 (1995), pp. 1399–1410. DOI: 10.1109/4.482187.

[20] Jianzhong Qian et al. “Microwave FM receiver using zero intermediate frequency”. In: *ICMMT’98. 1998 International Conference on Microwave and Millimeter Wave Technology. Proceedings (Cat. No.98EX106)*. 1998, pp. 181–184. DOI: 10.1109/ICMMT.1998.768255.

[21] Jan Thoemel and Eliott Hubin. “Personal communication”. Phone call. Mar. 2023.

[22] Curtis Barrett. *Fractional/Integer-N PLL Basics*. Tech. rep. Texas Instrument, Aug. 1999.

[23] Martin Ummenhofer et al. “Direction of Arrival Estimation Techniques for Passive Radar based 3D Target Localization”. In: *2019 IEEE Radar Conference (RadarConf)*. 2019, pp. 1–6. DOI: 10.1109/RADAR.2019.8835841.

[24] *MUSIC (algorithm)*. May 2023. URL: https://en.wikipedia.org/wiki/MUSIC_(algorithm). (accessed: 03.05.2023).

[25] R. Schmidt. "Multiple emitter location and signal parameter estimation". In: *IEEE Transactions on Antennas and Propagation* 34.3 (1986), pp. 276–280. DOI: 10.1109/TAP.1986.1143830.

[26] Yuzhou Liu and Bin Zhao. "Phase-shift correlation method for accurate phase difference estimation in range finder." In: *Applied optics* 54 11 (2015), pp. 3470–7.

[27] I. Kostiukov. "Decreasing of Bias of Modified Cross-Correlation Method for Phase Shift Measurement". In: *2022 IEEE 3rd KhPI Week on Advanced Technology (KhPIWeek)*. 2022, pp. 1–6. DOI: 10.1109/KhPIWeek57572.2022.9916443.

[28] Greg Hislop and Christophe Craeye. "On the Mathematical Link Between the MUSIC Algorithm and Interferometric Imaging". In: *IEEE Transactions on Antennas and Propagation* 59.4 (2011), pp. 1412–1414. DOI: 10.1109/TAP.2011.2109691.

[29] Mohammad K. Azarian. "Meftab Al-Hesab: A Summary". In: *Missouri Journal of Mathematical Sciences* 12.2 (2000), pp. 75–95. DOI: 10.35834/2000/1202075. URL: https://doi.org/10.35834/2000/1202075.

[30] David Gaidioz et al. "28-nm FD-SOI CMOS Submilliwatt Ring Oscillator-Based Dual-Loop Integer-N PLL for 2.4-GHz Internet-of-Things Applications". In: *IEEE Transactions on Microwave Theory and Techniques* 70.4 (2022), pp. 2207–2216. DOI: 10.1109/TMTT.2022.3149826.

Appendices

A Antenna layouts

A.1 Result parameters

In order to obtain the results shown in Section 3.4, the parameters given in Table 2 need to be used.

# Fig.	$\rho_{\min}$	$\rho_{\max}$	$\psi_{\min}$	$\psi_{\max}$
8	5	47	$\pi/5.0$	$\pi/4.0$
10	2	8	$\pi/6.0$	$\pi/5.1$

Table 2: Layout parameters

A.2 Codes

Listing 1: main.py

```
import matplotlib.pyplot as plt
from Sats_layout import *
from Radiation import *
#from Sats_layout_GRS import *
from Spirals_layout import *
from Utils import *
from Optimize import *
import numpy as np


##########Data#########
center_array_sats=1000*np.array([6771*np.cos(np.pi/2)*np.
    sin(0),6771*np.cos(np.pi/2)*np.cos(0),6771*np.sin(np.
    pi/2)]) # coordinates of the center of the antenna
    array (here : [0,0,6771]km)
ratio_angle = 7500 * np.pi /180 #km per degree of lat/
    long
ratio_dist = 1 /ratio_angle #degree of lat_long
    equivalent to 1km
specular_cart_coord=1000*np.array([6371*np.cos(np.pi/2)*
    np.sin(0),6371*np.cos(np.pi/2)*np.cos(0),6371*np.sin(
    np.pi/2)]) #cartesian coordinates of the specular
    point (here : [0,0,6371]km)
```

```
16 u0 = (center_array_sats - specular_cart_coord)/np.linalg.
       norm(center_array_sats - specular_cart_coord) #
       direction of interest for beam steering method
17 lambd = 0.2 #wavelength [m]
18 width = 50 #width for ground spot computing
19 glistening_width = 10 #km
20 theta_cut = 90 #choses the cut
21
22 ##### Decides if searching a layout or testing one
23
24 optimize = False
25
26 #### Decides if computing complete F and D or partial F
27
28 complete = False
29
30 ##Plot and computation activation##
31 Layout = 0
32 Pattern = 1
33 Directivity = 0
34 GroundSpot = 0
35
36 ##### Radiation pattern and directivity computations
       #####
37 if (Directivity or Pattern):
38     alpha = compute_glistening_angle(glistening_width)
39     if (optimize):
40         acc = 0.01
41         find_optimal_layout(N_arm, N_per_arm,
               center_array_sats, lambd, acc, theta_cut,
               glistening_width, p)
42     else:
43         ##### Set antenna's pattern #####
44         N_arm = 4
45         N_per_arm = 7
46         #ant_pos = GRSLayout(4,5,center_array_sats)
47         ant_pos = SpiralsLayout(N_arm,N_per_arm,
               center_array_sats,lambd,0,2,8,np.pi/6,np.pi
               /5.1)
48         #ant_pos = random_layout(80, 29)
```

```
49
50          smallest_distance = compute_distances(ant_pos)
51          print("The smallest distance between 2 CubeSats
                is : {:.2f} m".format(smallest_distance))
52
53          if (complete) :
54              acc = 0.1 # [deg] accuracy for the radiation
                    pattern and directivity
55              t = np.arange(0,180,acc)
56              p = np.arange(0,360,acc)
57              theta = t*np.pi/180 #deg to rad
58              phi = p*np.pi/180 #deg to rad
59              index_theta = int(theta_cut/acc) #choice of
                    the cut for F and D
60
61              F,D,integ = ray_diag(ant_pos,acc,lambd)
62              D = 10*np.log10(D)
63              print("D = {:.2f} dBi".format(np.amax(D)))
64          else:
65              acc = 0.01 # [deg] accuracy for the radiation
                    pattern
66              phi_min = 170
67              phi_max = 190
68              p = np.arange(phi_min,phi_max+acc,acc)
69              F = ray_diag_uncomplete(ant_pos, lambd,p,acc)
70
71
72 ############################
73          #Plots
74 ############################
75
76 ####Some check variables###
77 spiral = False #The layout is formed of spiral or not
78 save = False #To save or not figures
79
80 ##Antennas layout##
81 if (Layout):
82      plt.figure()
83      if (spiral):
84          for i in range(N_arm):
```

```
85                plt.plot(ant_pos[i*N_per_arm:(i+1)*N_per_arm
                      ,0],ant_pos[i*N_per_arm:(i+1)*N_per_arm
                      ,1],linestyle="--",marker="o",markersize
                      =5)
86            plt.scatter(ant_pos[-1,0],ant_pos[-1,1],color="
                  black")
87        else:
88            plt.scatter(ant_pos[:,0],ant_pos[:,1],s=5)
89
90        plt.xlabel("X axis [m]")
91        plt.ylabel("Y axis [m]")
92        plt.title("Antennas layout in X-Y plan (" r'$\lambda$
              ' "= {} m)".format(lambd) )
93        plt.grid()
94        plt.axis("square")
95        if (save):
96            plt.savefig("Layout_211_antennas_r_330.svg")
97
98    ##Radiation pattern##
99    if(Pattern):
100       if (complete):
101           fig, ax = plt.subplots(subplot_kw={'projection':
                  'polar'})
102           ax.plot(phi,F[:,index_theta])
103           ax.set_title(r"Radiation pattern for " r'$\theta$
                  ' "= {}$^\circ$".format(int(index_theta*acc))
                  )
104           ax.set_xlabel(r'$\phi$')
105           if (save):
106               plt.savefig("Pattern_29_antennas_5.svg")
107       else:
108           plt.figure()
109           plt.plot(p,10*np.log10(F[:,0]),'b')
110           plt.axvline(x=180+alpha,color="green",linestyle='
                  --',label="Glistening zone")
111           plt.axvline(x=180-alpha,color="green",linestyle='
                  --')
112           plt.ylim(-15,0)
113           plt.xlim(178,182)
114           plt.grid()
```

```
115             plt.title(r"Radiation␣pattern␣for␣"r'$\theta$'   "
                    =␣{}$^\circ$".format(theta_cut))
116             plt.xlabel(r'$\phi$' "␣[$^\circ$]")
117             plt.ylabel("AF␣[dB]")
118             plt.legend(loc="upper␣right")
119             if (save):
120                 plt.savefig("AF_Esp_300.svg")
121
122 ##Directivity##
123 if(Directivity):
124     if (complete):
125             plt.figure()
126             plt.plot(p,D[:,index_theta],'b')
127             plt.axvline(x=180+alpha,color="green",linestyle='
                    --',label="Glistening␣zone")
128             plt.axvline(x=180-alpha,color="green",linestyle='
                    --')
129             plt.xlim(178,182)
130             #plt.xlim(88,92)
131             plt.ylim(-15,15)
132             plt.grid()
133             plt.title(r"Directivity␣for␣"r'$\theta$'   "=␣{}$
                    ^\circ$".format(int(index_theta*acc)))
134             plt.xlabel(r'$\phi$' "␣[$^\circ$]")
135             plt.ylabel("Directivity␣[dB]")
136             plt.legend(loc="upper␣right")
137             if (save):
138                 plt.savefig("Directivity_211_antennas_r_330.
                        svg")
139
140 ##Ground spot##
141 if(GroundSpot):
142     #w = 0.1
143     #ground_spot_1(D[900-int(w*10):900+int(w*10)+1,900-
            int(w*10):900+int(w*10)+1],w)
144     w = width
145     n = 101
146     G = ground_spot(center_array_sats,specular_cart_coord
            ,u0,integ, ant_pos,w,n)
147     x = np.linspace(-w*10,w*10,n)
```

```
148     y = np.linspace(-w*10,w*10,n)
149     X,Y = np.meshgrid(x,y)
150     plt.figure()
151     CS = plt.contourf(X,Y,G,10,cmap="viridis")
152     plt.colorbar()
153     plt.grid()
154     plt.axis("square")
155     axes = plt.gca()
156     plt.xlabel("[m]")
157     plt.ylabel("[m]")
158     plt.title("Ground spot for {} antennas".format(
            ant_pos.shape[0]))
159     if (save):
160         plt.savefig("Ground_Spot_29_antennas_5.png")
161
162 ##Resolution##
163 try:
164     if (Directivity):
165         if(complete):
166             res = compute_resolution(D,index_theta,acc)
167         else:
168             res = compute_resolution_partial(F, acc,
                        phi_min, phi_max)
169         print("The resolution is equal to {:.2f} km".
                format(res))
170 except:
171     print("Resolution Computation failed")
172
173 plt.show()
```

Listing 2: Radiation.py

```
1 import numpy as np
2 from numba import jit
3 from matplotlib import pyplot as plt
4 from Utils import *
5
6 @jit(nopython=True)
7 def ray_diag(antenna_pos,acc,lambd):
8     """
9     Parameters
```

```
10      ----------
11      antenna_pos : Numpy array with dimensions : N x 3
12          Contains the cartesian coordinates of the N
                antenna's.
13      acc : int or float
14          accuracy of the computation, e.g. : 1 deg or 0.1
                deg.
15
16      Returns
17      -------
18      F : Numpy array with dimensions : m x n
19          Radiation Pattern.
20      D : Numpy array with dimensions : m x n
21          Directivity.
22      integral : int
23          integral of F over all phi and theta.
24      """
25      m = int(360/acc)
26      n = int(180/acc)
27      #lambd = 3e8/2.472e9
28      k = 2*np.pi/lambd
29      u = np.zeros((m,n,3)) #on va creer un u(x,y,z) pour
            tous les (phi, theta)
30      F = np.zeros((m,n),dtype="complex") #chaque (phi,
            theta) aura sa valeur de F
31      for i in range(m): #boucle sur les phi
32          for l in range(n): #boucle sur les theta
33              u[i,l,:] = [np.cos(l*np.pi/n)*np.sin(i*np.pi/
                    n),np.sin(l*np.pi/n)*np.sin(i*np.pi/n),np.
                    cos(i*np.pi/n)] #unit director vector u
34              for j in range(antenna_pos.shape[0]): #boucle
                    sur les antennes/satellites
35                  F[i,l] = F[i,l] + np.exp(1j*k*np.dot(u[i,
                        l,:],antenna_pos[j,:])) #*np.exp(-1*1j
                        *k*np.dot(u0,antenna_pos[n,:]))
36
37      F = np.abs(F)
38      F = F/np.amax(F)
39      t = np.deg2rad(np.arange(0,180,acc))
40      p = np.deg2rad(np.arange(0,360,acc))
```

```
41      sin_theta = np.sin(t)
42      integral = 0
43      for i in range(n):
44          for j in range(m):
45              integral = integral + (np.pi/n)*(2*np.pi/m) *
                    F[j,i]**2 * sin_theta[i]
46      D = (4*np.pi/integral)*(F**2)
47      return F,D,integral
48
49
50  def ray_diag_uncomplete(antenna_pos,lambd,p,acc):
51
52      k = 2*np.pi/lambd
53      t = 90
54      m = int(p.shape[0])
55      n = int(180/acc)
56      u = np.zeros((m,1,3))
57      F = np.zeros((m,1),dtype="complex")
58      for i in range(m): #boucle sur les phi
59          phi = p[i]
60          u[i,0,:] = [0,np.sin(phi*np.pi/180),np.cos(phi*np
                .pi/180)] #unit director vector u
61          for j in range(antenna_pos.shape[0]): #boucle sur
                les antennes/satellites
62              F[i,0] = F[i,0] + np.exp(1j*k*np.dot(u[i
                    ,0,:],antenna_pos[j,:]))
63      F = np.abs(F)
64      F = F/np.amax(F)
65
66      return F
```

Listing 3: Spirals_layout.py

```
1  import numpy as np
2  from matplotlib import pyplot as plt
3  from Utils import *
4
5  def SpiralsLayout(N_arm,N_per_arm,center_array_sats,lambd
       ,DispFigEn,rho_min,rho_max,psy_min,psy_max):
6
7       """
```

```
8       x_n = x_n-1 + rho_n*cos(phi_n-1 + psy_n)
9       y_n = y_n-1 + rho_n*sin(phi_n-1 + psy_n)
10
11      with phi_n = phi_n-1 + psy_n
12      """
13      ######################
14      #####Parameters#####
15      ######################
16      #Rmin = 13 #circle's radius made by the first
            antennas of each arm if one wants to adjust it
17      if (rho_max <= rho_min):
18          print("Invalid rho's : rho_min = {} and rho_max =
                {}".format(rho_min, rho_max))
19
20      ################################
21      #####Spirals construction#####
22      ################################
23      #center_array_sats = np.array([0,6771*1000,0])
24      Rmin = rho_min
25      rho = np.linspace(Rmin,rho_max,N_per_arm) #successive
            rho's between antenna's in an arm
26      #rho = np.linspace(Rmin,rho_max,N_per_arm-1) #if Rmin
            is different from rho_min
27
28      psy = np.linspace(psy_min,psy_max,N_per_arm-1) #
            successive psy's between antenna's in an arm
29      phi = np.zeros((N_arm,N_per_arm-1)) #successive phi's
            between antenna's in an arm
30      phi[0,0] = psy[0] #First phi of the first arm
31
32      for n in range(1,N_per_arm-1): #Instantiation of all
            the phi's of the first arm
33          phi[0,n] = phi[0,n-1] + psy[n]
34
35      for m in range(1,N_arm):    #Computation of the phi's
            for the other arms
36          phi[m] = phi[m-1] + 2*np.pi/N_arm
37
38
39      theta_0 = np.linspace(0,2*np.pi,N_arm,endpoint=False)
```

```
        #Disposition of the first antenna's of each arm
    x = np.zeros((N_arm,N_per_arm))
    y = np.zeros((N_arm,N_per_arm))
    x[:,0] = Rmin*np.cos(theta_0)
    y[:,0] = Rmin*np.sin(theta_0)

    for m in range(N_arm):
        for n in range(1,N_per_arm):
            rho_n = rho[n]
            #rho_n = rho[n-1] #if Rmin is different from
                rho_min
            phi_n = phi[m,n-1]
            x[m,n] = x[m,n-1] + rho_n*np.cos(phi_n)
            y[m,n] = y[m,n-1] + rho_n*np.sin(phi_n)

    GRS = np.array([np.ravel(x),np.ravel(y),np.zeros((
        N_arm*N_per_arm))])
    newGRS = np.zeros((N_arm*N_per_arm,3))
    for j in range(0,N_arm*N_per_arm):
        newGRS[j][:] = [GRS[0][j],GRS[1][j],GRS[2][j]] +
            center_array_sats
        #newGRS[j][:] = [GRS[0][j],GRS[2][j],GRS[1][j]] +
            center_array_sats

    newGRS = np.append(newGRS,[center_array_sats],axis=0)

    if (DispFigEn):
        plt.figure()
        plt.scatter(x[0,:],y[0,:])
        plt.grid()
        plt.axis("square")
        plt.title("One arm layout")
        #plt.show()

    return newGRS

def GRSLayout(N_arm,N_per_arm,center_array_sats):
    Density = 1
    Rmin = 0.25
    Rmax = np.sqrt(N_arm*N_per_arm/(np.pi*Density))
```

```
75      theta = np.zeros((N_arm,N_per_arm))
76      #radii = np.linspace(Rmin,Rmax,N_per_arm)
77      radii = np.geomspace(Rmin,Rmax,N_per_arm)
78      theta[0,:] = (np.pi/2)*np.log(radii)
79
80      delta = np.zeros((N_per_arm-1,1))
81      golden = (1 + 5 ** 0.5) / 2
82
83      ##Distance between antennas on smallest radius##
84      P = 2*np.pi*Rmin
85      Dist = P/N_arm
86
87      for i in range(1,N_arm):
88          theta[i,:] = theta[0,:] + (i)*2*np.pi/N_arm
89
90      #r = np.exp(2*theta[0,:]/np.pi) #exponential spiral
91      #r = golden**(2*theta[0,:]/np.pi)  #golden ratio
            spiral
92      r = 4.25*theta[0,:] #Archimedean spiral
93      #r = 4.5*(1/theta[0,:]) #Hyperbolic spiral
94
95      for ii in range(0,N_per_arm-1):
96          delta[ii] = ((r[ii+1]*np.cos(theta[0,ii+1])-r[ii
                ]*np.cos(theta[0,ii]))**2 + (r[ii+1])*np.sin(
                theta[0,ii+1])-r[ii]*np.sin(theta[0,ii]))
97
98      GRS = np.array([np.ravel(r.T*np.cos(theta)),np.ravel(
            r.T*np.sin(theta)),np.zeros((N_arm*N_per_arm))])
99      newGRS = np.zeros((N_arm*N_per_arm,3))
100     for j in range(0,N_arm*N_per_arm):
101         newGRS[j][:] = [GRS[0][j],GRS[1][j],GRS[2][j]] +
                center_array_sats
102
103     return newGRS
```

Listing 4: Optimize.py

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from Radiation import *
4 from Spirals_layout import *
```

```
5 from scipy.signal import find_peaks
6
7 def find_optimal_layout(N_arm,N_per_arm,center_array_sats
      ,lambd,acc,theta_cut,width,p):
8     index_theta = int(theta_cut/acc)
9     while (True):
10        rho_min = np.random.randint(5,15)
11        rho_max = np.random.randint(50,51)
12        psy_min_d = np.random.randint(50,80)/10
13        psy_min = np.pi/psy_min_d
14        psy_max_d = np.random.randint(35,psy_min_d*10)/10
15        psy_max = np.pi/psy_max_d
16        print(rho_min)
17        print(rho_max)
18        print(psy_min_d)
19        print(psy_max_d)
20        #ant_pos = SpiralsLayout(N_arm,N_per_arm,
              center_array_sats,lambd,0,rho_min,rho_max,
              psy_min,psy_max)
21        ant_pos = SpiralsLayout(N_arm,N_per_arm,
              center_array_sats,lambd,0,rho_min,rho_max,
              psy_min,psy_max)
22        plt.figure()
23        for i in range(N_arm):
24            plt.plot(ant_pos[i*N_per_arm:(i+1)*N_per_arm
                  ,0],ant_pos[i*N_per_arm:(i+1)*N_per_arm
                  ,1],linestyle="--",marker="o")
25        plt.scatter(ant_pos[-1,0],ant_pos[-1,1],color="
              black")
26        plt.xlabel("X axis [m]")
27        plt.ylabel("Y axis [m]")
28        plt.title("Antennas layout in X-Y plan (" r'$\
              lambda$' "= 0.2 m)" )
29        plt.grid(markevery=0.5)
30        plt.axis("square")
31
32        #F,D,integ = ray_diag(ant_pos,acc)
33        F,D,integ = ray_diag_uncomplete(ant_pos, lambd,
              0.4)
34        F = 10*np.log10(F)
```

```
35          #D = 10*np.log10(D)
36          print("D = {:.2f} dBi".format(np.amax(D)))
37          """
38          try:
39              res = compute_resolution(D, index_theta, acc)
40              print("The resolution is equal to {:.2f} km".
                    format(res))
41          except:
42              continue
43          """
44
45          alpha = compute_glistening_angle(width)
46          #peaks, _ = find_peaks(D[1792:1809, index_theta],
                height=-10)
47          peaks, _ = find_peaks(F[920:975,0], height=-6)
48          #peaks = np.delete(peaks, np.where(peaks == 8))
49          peaks = np.delete(peaks, np.where(peaks == 80))
50          print(peaks)
51          plt.figure()
52          #plt.plot(p,D[:, index_theta], 'b')
53          acc = 0.01
54          p = np.arange(170,190+acc, acc)
55          plt.plot(p,F[:,0], 'b')
56          #plt.plot((peaks+1792)/10, D[:, index_theta][peaks
                +1792], "x", color = "red", label="peaks")
57          #plt.plot(p,D[:,0], 'b')
58          plt.axvline(x=180+alpha, color="green", linestyle='
                --', label="Glistening zone")
59          plt.axvline(x=180-alpha, color="green", linestyle='
                --')
60          #plt.xlim(90,270)
61          plt.xlim(179,181)
62          #plt.ylim(-15,15)
63          plt.ylim(-15,0)
64          plt.grid()
65          plt.title("Directivity for "r'$\theta$'  "= {}$^\
                circ$".format(int(index_theta*acc)))
66          plt.xlabel(r'$\phi$' " [$^\circ$]")
67          plt.ylabel("Directivity [dB]")
68          plt.legend()
```

```
69          plt.show()
70          if(peaks.shape[0]==0):
71              print("Optimization has ended")
72              break
```

Listing 5: Utils.py

```
1  import numpy as np
2  from numba import jit
3  import matplotlib.pyplot as plt
4  from itertools import combinations
5
6  def compute_resolution(D,index_theta,acc):
7
8      x = np.arange(0,360,0.0001)
9      xp = np.arange(0,360,acc)
10     y = np.interp(x,xp,D[:,index_theta])
11     yf = np.argwhere((y[:] < (D[int(180/acc),index_theta]
           -3)) & (y[:] > (D[int(180/acc),index_theta]-3.05)
       ))
12     yf1 = np.extract((yf>1800000) & (yf<1900000),yf)
13     angle = (yf1[0]/1000)-180
14     x = 400000*np.tan(np.deg2rad(2*angle))
15
16     return x/1000
17
18 def compute_resolution_partial(F,acc,phi_min,phi_max):
19     dx = 0.0001
20     F = 10*np.log10(F)
21     x = np.linspace(phi_min,phi_max,int((F.shape[0]-1)/dx
       )+1)
22     xp = np.arange(phi_min,phi_max+acc,acc)
23     y = np.interp(x,xp,F[:,0])
24     idx = int(((phi_max - phi_min)/2)/(acc))
25     yf = np.argwhere((y[:] < (F[idx,0] -3)) & (y[:] > (F[
       idx,0]-3.001)))
26     yf1 = np.extract(yf>idx/dx,yf)
27     angle = x[yf1[0]]-180
28     x = 400000*np.tan(np.deg2rad(2*angle))
29
30     return x/1000
```

```
31
32
33
34
35 @jit(nopython=True)
36 def ground_spot_2(specular_coord, sat_center_coord,
       sat_disp, zone_area, integral):
37     """
38     This is the translation of the Ugo-Paul's Matlab code
39     """
40     width = zone_area*10 #precision = 100m
41     #width = 1*10
42     F = np.zeros((width+1,width+1),dtype="complex") #
           Radiation pattern
43     lambd = 0.2
44     k = 2*np.pi/lambd
45
46     u0 = (sat_center_coord - specular_coord)/np.linalg.
          norm(sat_center_coord - specular_coord) #beam
          steering method, u0: vector pointing from the
          center of the array to the specular point
47     u0 = -1*u0
48     ground_coord = np.zeros((width+1,width+1,3))
49     u = np.zeros((width+1,width+1,3))
50
51     for i in range(width+1):
52         for l in range(width+1):
53             ground_coord[i,l,0] = specular_coord[0] + (i
                  - (width/2+1))
54             ground_coord[i,l,1] = specular_coord[1] + (l
                  - (width/2+1))
55             ground_coord[i,l,2] = specular_coord[2] + 0
56
57             u[i,l,0] = ground_coord[i,l,0] -
                  sat_center_coord[0]
58             u[i,l,1] = ground_coord[i,l,1] -
                  sat_center_coord[1]
59             u[i,l,2] = ground_coord[i,l,2] -
                  sat_center_coord[2]
60
```

```
61                for n in range(sat_disp.shape[0]):
62                    rho = sat_disp[n,:] - sat_center_coord
63                    F[i,l] = F[i,l] + np.exp(1j*k*np.dot(u[i,
                          l,:]/np.linalg.norm(u[i,l,:]),rho))#*
                          np.exp(-1j*k*np.dot(u0,rho)) #
                          beamsteering method
64
65
66        #integ = np.trapz(np.trapz(np.abs(F)**2,dx=0.01,axis
              =1),dx=0.01,axis=0)
67        #print(integ)
68        F = np.abs(F)
69        F = F/np.amax(F)
70        D = (4*np.pi/integral)*(F**2)
71        #D = (4*np.pi/integ)*(F**2)
72        D = 10*np.log10(D)
73        G = 1*D
74
75        return G,F
76
77    def ground_spot_1(D,w):
78        """
79        This is a try of making a ground spot plot directly
              with the directivity computed in spherical
              coordinates
80        """
81        x = 400*np.tan(np.deg2rad(np.linspace(-w,w,D.shape
              [1])))
82        y = 400*np.tan(np.deg2rad(np.linspace(-w,w,D.shape
              [0])))
83        X,Y = np.meshgrid(x,y)
84        plt.figure()
85        plt.contourf(Y,X,D,10,cmap="viridis")
86        plt.colorbar()
87        plt.grid()
88        plt.axis("square")
89        plt.xlabel("[km]")
90        plt.ylabel("[km]")
91        return
92
```

```
@jit(nopython=True)
def ground_spot(sat_center_coord,specular_coord,u0,integ,
    sat_disp,w,n):
    lambd = 0.2
    k = 2*np.pi/lambd
    x = np.linspace(-w,w,n)
    y = np.linspace(-w,w,n)
    ground_coord = np.zeros((len(x),len(y),3))
    F = np.zeros((len(x),len(y)),dtype="complex")
    u = np.zeros((len(x),len(y),3))
    u0 = -1*u0
    for i in range(len(x)):
        for j in range(len(y)):
            ground_coord[i,j,0] = x[i]
            ground_coord[i,j,1] = y[j]
            ground_coord[i,j,2] = 6371*1000

            u[i,j,0] = ground_coord[i,j,0] -
                sat_center_coord[0]
            u[i,j,1] = ground_coord[i,j,1] -
                sat_center_coord[1]
            u[i,j,2] = ground_coord[i,j,2] -
                sat_center_coord[2]

            u_n = u[i,j,:]/np.sqrt(u[i,j,0]**2+u[i,j
                ,1]**2+u[i,j,2]**2)

            for n in range(sat_disp.shape[0]):
                rho = sat_disp[n,:] - sat_center_coord
                F[i,j] = F[i,j] + np.exp(1j*k*np.dot(u_n,
                    rho)) #*np.exp(-1j*k*np.dot(u0,rho))

    F = np.abs(F)
    F = F/np.amax(F)
    D = (4*np.pi/integ)*(F**2)
    D = 10*np.log10(D)

    return D


```

```
127 def compute_radius(lambd,res):
128     alpha = 0.5*np.arctan(res/400)
129     b = lambd/(2*np.sin(alpha))
130     return b
131 
132 def compute_glistening_angle(x):
133     """
134     x = [km] glistening zone's width
135     """
136     alpha = 0.5*np.arctan(x/400)
137     return np.rad2deg(alpha)
138 
139 def compute_distances(points):
140     distances = []
141     for p1, p2 in combinations(points, 2):
142         distance = np.linalg.norm(p2 - p1)
143         distances.append(distance)
144     return min(distances)
```

B Processing

Listing 6: CorrMat.py

```
1  """
2  Note : This code is inspired by a MATLAB code provided by
       Maxime Drouguet
3  """
4  import numpy as np
5  from matplotlib import pyplot as plt
6 
7  def CorrMat(data, recalibration):
8      """
9      LOCALISATION Summary of this function goes here
10     Detailed explanation goes here
11     """
12 
13     n_ant = data.shape[0]
14     n_samp = data.shape[1]
15 
16     Complex = data.T
17 
```

```
18      if (recalibration > 0):
19          cor = np.zeros((n_ant,n_ant))
20          for n in range(n_samp):
21              cor = cor + np.reshape((Complex[n,:].T),(
                    n_ant,1))@np.reshape(np.conj(Complex[n,:])
                    ,(1,n_ant))
22
23          cor = cor/n_samp
24          np.save("CorrMat",cor)
25
26      cor = np.load("CorrMat.npy")
27      for i in range(n_ant):
28          Complex[:,i] = Complex[:,i]/(np.conj(cor[0,i]).T)
29
30
31      #-----------------------------------------
32      #-Calcul de la matrice de correlation
33      #-----------------------------------------
34      cor = np.zeros((n_ant,n_ant))
35      for n in range(n_samp):
36              cor = cor + np.reshape((Complex[n,:].T),(
                    n_ant,1))@np.reshape(np.conj(Complex[n,:])
                    ,(1,n_ant))
37      cor = cor/n_samp
38
39      return cor
```

Listing 7: MUSIC.py

```
1  import numpy as np
2  from matplotlib import pyplot as plt
3
4  def MUSIC(corr_mat,posx,posy,lambd):
5      """
6      corr_mat = correlation matrix
7      posx,posy = vectors containing the antennas
           coordinates
8
9      """
10     k = 2*np.pi/lambd
11     n_ant = corr_mat.shape[0] #number of antennas
```

```
12
13      Di,Ei = np.linalg.eig(corr_mat) #eigen decomposition
            note Ei are eigen vectors Di eigen values
14
15      Un = np.reshape(Ei[:,np.argmin(Di)],(2,1)) #The noise
            space is the eigen vector that corresponds to the
            lowest eigen value
16      Un_H = Un.conj().T
17
18      theta = np.arange(0,180,1) #swipe on all directions
19      a = np.zeros((n_ant,theta.shape[0]),dtype="complex")
            #steering vector
20      pow = np.zeros((theta.shape[0],1),dtype="complex")
21
22      for i in range(theta.shape[0]):
23          t = np.deg2rad(theta[i])
24          a[:,i] = np.exp(1j*k*posx*np.cos(t))
25          a_H = a[:,i].conj().T
26          pow[i] = 1/(a_H@Un@Un_H@a[:,i])
27
28      powdb = 10*np.log10(np.abs(pow))
29
30      return powdb
```

C GUI

Listing 8: main.py

```
1 from Window import *
2
3 if __name__ == '__main__':
4     app = QApplication(sys.argv)
5
6     main = MainWindow()
7     main.show()
8
9     sys.exit(app.exec_())
```

Listing 9: window.py

```
1 import sys
```

```
2 import matplotlib
3 matplotlib.use('Qt5Agg')
4 from matplotlib.backends.backend_qt5agg import
      FigureCanvasQTAgg, NavigationToolbar2QT as
      NavigationToolbar
5 from matplotlib.figure import Figure
6
7 import random
8 import time
9
10 from PyQt5.QtGui import *
11 from PyQt5.QtWidgets import *
12 from PyQt5.QtCore import *
13
14 from Acquisition import *
15 from Processing import *
16 from ESP300 import *
17
18 global UDP0,UDP1
19
20 class MplCanvas(FigureCanvasQTAgg):
21
22     def __init__(self, parent=None, width=5, height=4,
          dpi=100,xlabel = "",ylabel="",title="Title"):
23         fig = Figure(figsize=(width, height), dpi=dpi)
24         self.axes = fig.add_subplot(111)
25         self.axes.set_xlabel(xlabel)
26         self.axes.set_ylabel(ylabel)
27         self.axes.set_title(title)
28         self.axes.grid()
29         super(MplCanvas, self).__init__(fig)
30
31 class Worker(QRunnable):
32     '''
33     Worker thread
34     '''
35
36     @pyqtSlot()
37     def run(self):
38         global UDP0,UDP1
```

```
39          '''
40          Your code goes in this function
41          '''
42          print("Thread start")
43          UDP0,UDP1 = acquisition()
44          print("Thread complete")
45
46  class MainWindow(QMainWindow):
47
48      def __init__(self):
49          super(MainWindow, self).__init__()
50
51          self.setWindowTitle("Acquisition GUI")
52          self.setMinimumSize(QSize(1200, 600))
53
54          layout1 = QHBoxLayout()
55          layout2 = QVBoxLayout()
56          layout21 = QHBoxLayout()
57          layout22 = QVBoxLayout()
58          layout23 = QVBoxLayout()
59          layout24 = QVBoxLayout()
60          layout3 = QVBoxLayout() #right column
61          layout31 = QHBoxLayout() #Above the plot
62          layout32 = QVBoxLayout() #The plot and its
                toolbar
63          layout311 = QVBoxLayout() #Left column of
                layout31
64          layout312 = QVBoxLayout() #Central column of
                layout31
65          layout313 = QVBoxLayout() #Right column of
                Layout31
66
67
68          layout1.setContentsMargins(20,20,20,20)
69          layout1.setSpacing(20)
70
71          #RX0 position input box
72          Box1 = QDoubleSpinBox()
73          Box1.setMinimum(-1)
74          Box1.setMaximum(1)
```

```
75          Box1.setValue(-0.15)
76          Box1.setPrefix("Pos RX0 = ")
77          Box1.setSuffix(" [m]")
78          Box1.setSingleStep(0.01)  # Or e.g. 0.5 for
                QDoubleSpinBox
79          Box1.valueChanged.connect(self.value_changedRX0)
80          Box1.textChanged.connect(self.value_changed_str)
81          layout22.addWidget(Box1)
82
83          #RX1 position input box
84          Box2 = QDoubleSpinBox()
85          Box2.setMinimum(-1)
86          Box2.setMaximum(1)
87          Box2.setValue(0.15)
88          Box2.setPrefix("Pos RX1 = ")
89          Box2.setSuffix(" [m]")
90          Box2.setSingleStep(0.01)  # Or e.g. 0.5 for
                QDoubleSpinBox
91          Box2.valueChanged.connect(self.value_changedRX1)
92          Box2.textChanged.connect(self.value_changed_str)
93          layout22.addWidget(Box2)
94
95          #TX2472 position input box
96          Box3 = QDoubleSpinBox()
97          Box3.setMinimum(0)
98          Box3.setMaximum(2)
99          Box3.setPrefix("Pos TX2472 = ")
100         Box3.setSuffix(" [m]")
101         Box3.setSingleStep(0.05)  # Or e.g. 0.5 for
                QDoubleSpinBox
102         Box3.valueChanged.connect(self.value_changedDoA)
103         Box3.textChanged.connect(self.value_changed_str)
104         layout23.addWidget(Box3)
105
106         #Filename input box
107         Line = QLineEdit()
108         Line.setPlaceholderText("DefaultName.npy")
109         Line.textChanged.connect(self.text_changed)
110         Line.textEdited.connect(self.text_edited)
111         layout23.addWidget(Line)
```

```
112
113             #TX434 - RX0 initial distance
114             Box4 = QDoubleSpinBox()
115             Box4.setMinimum(0)
116             Box4.setMaximum(5)
117             Box4.setPrefix("Dist TX434-RX0 = ")
118             Box4.setSuffix(" [m]")
119             Box4.setSingleStep(0.005)  # Or e.g. 0.5 for
                    QDoubleSpinBox
120             Box4.setValue(1.365)
121             Box4.valueChanged.connect(self.value_changedDist0
                    )
122             Box4.textChanged.connect(self.value_changed_str)
123             layout24.addWidget(Box4)
124
125             #TX434 - RX1 initial distance
126             Box5 = QDoubleSpinBox()
127             Box5.setMinimum(0)
128             Box5.setMaximum(5)
129             Box5.setPrefix("Dist TX434-RX1 = ")
130             Box5.setSuffix(" [m]")
131             Box5.setSingleStep(0.005)  # Or e.g. 0.5 for
                    QDoubleSpinBox
132             Box5.setValue(1.655)
133             Box5.valueChanged.connect(self.value_changedDist1
                    )
134             Box5.textChanged.connect(self.value_changed_str)
135             layout24.addWidget(Box5)
136
137             #CheckBox to correct or not the phase shift due
                    to RX displacement
138             Box6 = QCheckBox("Correct RX displacement")
139             Box6.setCheckState(Qt.Unchecked)
140             Box6.stateChanged.connect(self.show_state)
141             Box6.stateChanged.connect(self.update_flag)
142             layout311.addWidget(Box6)
143
144
145             self.lcd1 = QLCDNumber()
146             self.lcd1.display(0.0)
```

```
147             #self.lcd.resize(10,5)
148             layout311.addWidget(self.lcd1)
149 
150             self.ESP300_X = 0
151             self.ESP300_Y = 0
152 
153             #ESP 300 - X coordinate
154             self.Box6 = QDoubleSpinBox()
155             self.Box6.setMinimum(-250)
156             self.Box6.setMaximum(250)
157             self.Box6.setPrefix("ESP300 : X = ")
158             self.Box6.setSuffix(" [mm]")
159             self.Box6.setSingleStep(10)  # Or e.g. 0.5 for
                    QDoubleSpinBox
160             self.Box6.valueChanged.connect(self.
                    value_changed_ESP300_X)
161             self.Box6.textChanged.connect(self.
                    value_changed_str)
162             layout312.addWidget(self.Box6)
163 
164             """
165             #ESP 300 - Y coordinate
166             self.Box7 = QDoubleSpinBox()
167             self.Box7.setMinimum(-125)
168             self.Box7.setMaximum(125)
169             self.Box7.setPrefix("ESP300 : Y = ")
170             self.Box7.setSuffix(" [mm]")
171             self.Box7.setSingleStep(0.005)  # Or e.g. 0.5 for
                    QDoubleSpinBox
172             self.Box7.valueChanged.connect(self.
                    value_changed_ESP300_Y)
173             self.Box7.textChanged.connect(self.
                    value_changed_str)
174             layout312.addWidget(self.Box7)
175             """
176 
177             ComboBox = QComboBox()
178             ComboBox.addItems(["LOS_FLG","OOF_FLG","LOL_FLG",
                    "HOLD_FLG"])
179             ComboBox.currentIndexChanged.connect(self.
```

```
            index_changed)
180         layout312.addWidget(ComboBox)
181
182         self.flag_idx = 0
183
184         layout31.addLayout(layout311)
185         layout31.addLayout(layout312)
186         layout31.addLayout(layout313)
187
188         layout21.addLayout(layout22)
189         layout21.addLayout(layout23)
190         layout21.addLayout(layout24)
191         layout2.addLayout(layout21)
192
193         self.posx = np.array([-0.15,0.15])
194         self.DoA = 90
195         self.filename = "DefaultName.npy"
196         self.DistTX434_RX0 = 0
197         self.DistTX434_RX1 = 0
198         self.k = 2*np.pi/(3e8/434e6)
199
200         self.ESP300 = 0
201
202         #INITIAL angles and lengths
203         self.alpha = 0
204         self.beta = 0
205         self.gamma = 0
206
207         self.delta_dist = 0 #b'- b or c'- c
208
209         self.UpdatePhaseShiftFlag = 0
210
211         n_data = 250
212
213         #I component plot
214         self.plot1 = MplCanvas(self, width=5, height=4,
            dpi=100,title="I component")
215         self.xdata1 = list(range(n_data))
216         self.ydata10 = [random.randint(0, 10) for i in
            range(n_data)]
```

```
217         self.ydata11 = [random.randint(0, 10) for i in
                range(n_data)]
218         toolbar1 = NavigationToolbar(self.plot1, self)
219         layout2.addWidget(toolbar1)
220         layout2.addWidget(self.plot1)
221 
222         #Q component plot
223         self.plot2 = MplCanvas(self, width=5, height=4,
                dpi=100,title="Q component")
224         self.xdata2 = list(range(n_data))
225         self.ydata20 = [random.randint(0, 10) for i in
                range(n_data)]
226         self.ydata21 = [random.randint(0, 10) for i in
                range(n_data)]
227         toolbar2 = NavigationToolbar(self.plot2, self)
228         layout2.addWidget(toolbar2)
229         layout2.addWidget(self.plot2)
230 
231         layout1.addLayout( layout2 )
232 
233         #MUSIC plot
234         self.plot3 = MplCanvas(self, width=5, height=4,
                dpi=100,xlabel=r"Direction [$^\circ$]",ylabel=
                "Power [dB]",title="MUSIC power according to
                the direction of arrival")
235         self.xdata3 = list(range(180)) # X vector for
                MUSIC power
236         self.ydata30 = [random.randint(0, 10) for i in
                range(180)] # Y vector for MUSIC power
237         self.xdata30 = 0 # vertical line (DoA)
238         self.xdata31 = 50 # vertical line (DoA)
239         self.ydata31 = random.randint(15, 165) # vertical
                line (DoA)
240         toolbar3 = NavigationToolbar(self.plot3, self) #
                Plot toolbar
241 
242 
243         layout32.addWidget(toolbar3)
244         layout32.addWidget(self.plot3)
245 
```

```
246             layout3.addLayout(layout31)
247             layout3.addLayout(layout32)
248 
249             #Updating plots
250             self._plot_ref = [None,None,None,None,None,None]
251             self.update_plot()
252 
253             # Setup a timer to trigger the redraw by calling
                    update_plot.
254             self.timer = QTimer()
255             self.timer.setInterval(1000) #update every 1 sec
256             self.timer.timeout.connect(self.update_plot)
257             self.timer.start()
258 
259             layout1.addLayout(layout3)
260 
261             #Toolbar with the buttons
262             toolbar = QToolBar("My main toolbar")
263             self.addToolBar(toolbar)
264 
265             #Launch button
266             button_action = QAction("Launch", self)
267             button_action.setStatusTip("This is your first
                    button")
268             button_action.triggered.connect(self.
                    onMyToolBarButtonClick1)
269             toolbar.addAction(button_action)
270 
271             #Save button
272             button_action = QAction("Save", self)
273             button_action.setStatusTip("This is your second
                    button")
274             button_action.triggered.connect(self.
                    onMyToolBarButtonClick2)
275             toolbar.addAction(button_action)
276 
277             #Calib button
278             button_action = QAction("Calib", self)
279             button_action.setStatusTip("This is your third
                    button")
```

```
280             button_action.triggered.connect(self.
                    onMyToolBarButtonClick3)
281             toolbar.addAction(button_action)
282 
283             #End of acquisition button
284             button_action = QAction("End", self)
285             button_action.setStatusTip("This is your fourth
                    button")
286             button_action.triggered.connect(self.
                    onMyToolBarButtonClick4)
287             toolbar.addAction(button_action)
288 
289             #Clean UDP button
290             button_action = QAction("Clean UDP", self)
291             button_action.setStatusTip("This is your fifth
                    button")
292             button_action.triggered.connect(self.
                    onMyToolBarButtonClick5)
293             toolbar.addAction(button_action)
294 
295             #ESP300 Motor ON button
296             button_action = QAction("Motor ON", self)
297             button_action.setStatusTip("This is your sixth
                    button")
298             button_action.triggered.connect(self.
                    onMyToolBarButtonClick6)
299             toolbar.addAction(button_action)
300 
301             #ESP300 Motor OFF button
302             button_action = QAction("Motor OFF", self)
303             button_action.setStatusTip("This is your seventh
                    button")
304             button_action.triggered.connect(self.
                    onMyToolBarButtonClick7)
305             toolbar.addAction(button_action)
306 
307             #Requests and displays the applied phase shift
308             button_action = QAction("Query phase", self)
309             button_action.setStatusTip("This is your eighth
                    button")
```

```
310         button_action.triggered.connect(self.
                onMyToolBarButtonClick8)
311         toolbar.addAction(button_action)
312 
313         #Requests and displays the X position of ESP300
314         button_action = QAction("Query ESP-X", self)
315         button_action.setStatusTip("This is your ninth 
                button")
316         button_action.triggered.connect(self.
                onMyToolBarButtonClick9)
317         toolbar.addAction(button_action)
318 
319         """
320         #Requests and displays the Y position of ESP300
321         button_action = QAction("Query ESP-Y", self)
322         button_action.setStatusTip("This is your tenth 
                button")
323         button_action.triggered.connect(self.
                onMyToolBarButtonClick10)
324         toolbar.addAction(button_action)
325         """
326         #Moves the axis of ESP300
327         button_action = QAction("Move ESP", self)
328         button_action.setStatusTip("This is your eleventh 
                button")
329         button_action.triggered.connect(self.
                onMyToolBarButtonClick11)
330         toolbar.addAction(button_action)
331 
332         #Requests the flags of the PLL
333         button_action = QAction("Query flag", self)
334         button_action.setStatusTip("This is your twelvth 
                button")
335         button_action.triggered.connect(self.
                onMyToolBarButtonClick12)
336         toolbar.addAction(button_action)
337 
338         #Requests the flags of the PLL
339         button_action = QAction("Reset flags", self)
340         button_action.setStatusTip("This is your 
```

```
            thirteenth button")
341         button_action.triggered.connect(self.
            onMyToolBarButtonClick13)
342         toolbar.addAction(button_action)
343
344
345         #Central widget
346         widget = QWidget()
347         widget.setLayout(layout1)
348         self.setCentralWidget(widget)
349
350         #Threadpool to launch the acquisition in
            background
351         self.threadpool = QThreadPool()
352         print("Multithreading with maximum %d threads" %
            self.threadpool.maxThreadCount())
353
354
355     def value_changedRX0(self, i):
356         #Set the new position of RX0
357         self.posx[0] = i
358         if (self.UpdatePhaseShiftFlag):
359             a_p = np.abs(self.posx[0]) + np.abs(self.posx
                [1])
360             b = self.DistTX434_RX1
361             c = self.DistTX434_RX0
362             c_p = np.sqrt(a_p**2 + b**2 - 2*a_p*b*np.cos(
                self.gamma))
363             self.delta_dist = c_p - c
364         print(i)
365
366     def value_changedRX1(self, i):
367         #Set the new position of RX1
368         self.posx[1] = i
369         if (self.UpdatePhaseShiftFlag):
370             a_p = np.abs(self.posx[0]) + np.abs(self.posx
                [1])
371             b = self.DistTX434_RX1
372             c = self.DistTX434_RX0
373             b_p = np.sqrt(a_p**2 + c**2 - 2*a_p*c*np.cos(
```

```
                self.beta))
374             self.delta_dist = b_p - b
375         print(i)
376
377     def value_changedDoA(self, i):
378         #Set the new DoA
379         self.DoA = get_angle(i)
380         print(i)
381
382     def value_changedDist0(self, i):
383         #Set the new INITIAL distance between TX434 and
                RX0
384         self.DistTX434_RX0 = i
385         print(self.DistTX434_RX0)
386
387     def value_changedDist1(self, i):
388         #Set the new INITIAL distance between TX434 and
                RX1
389         self.DistTX434_RX1 = i
390         print(self.DistTX434_RX1)
391
392     def show_state(self, s):
393         print(s == Qt.Checked)
394         print(s)
395
396     def update_flag(self,s):
397         if (s==2):
398             self.UpdatePhaseShiftFlag = 1
399         else:
400             if(s==0):
401                 self.UpdatePhaseShiftFlag = 0
402         print("Phase shift flag is now at {}".format(self
                .UpdatePhaseShiftFlag))
403
404     def value_changed_str(self, s):
405         #Each time the value of QDoubleSpinBox is changed
406         print(s)
407
408     def text_changed(self, s):
409         #Set the new filename
```

```
410         self.filename = s
411         print("Text changed...")
412         print(self.filename)
413 
414     def text_edited(self, s):
415         #Executed when the text is changed
416         print("Text edited...")
417         print(s)
418 
419     def value_changed_ESP300_X(self,i):
420         #Sets the X position of ESP300
421         self.ESP300_X = i
422         #SetX(self.ESP300, i)
423         print(i)
424 
425     def value_changed_ESP300_Y(self,i):
426         #Sets the Y position of ESP300
427         self.ESP300_Y = i
428         #SetY(self.ESP300, i)
429         print(i)
430 
431     def index_changed(self,i):
432         #Changes the cuurent flag that we could request
                and display
433         print(i)
434         self.flag_idx = i
435 
436     def onMyToolBarButtonClick1(self, s):
437         #Button 1 : Launch the acquisition
438         print("click", s)
439         worker = Worker()
440         self.threadpool.start(worker)
441 
442     def onMyToolBarButtonClick2(self, s):
443         #Button 2 : Save the data
444         print("click", s)
445         IQ0,IQ1 = get_data()
446         Data = np.array([IQ0,IQ1])
447         Filename = self.filename
448         np.save(Filename, Data)
```

```
449         print("Data saved in file "+ Filename)
450
451     def onMyToolBarButtonClick3(self, s):
452         #Button 3 : Generates a calib matrix + saves the
                initial alpha
453         print("click", s)
454         a = np.abs(self.posx[0]) + np.abs(self.posx[1])
455         b = self.DistTX434_RX1
456         c = self.DistTX434_RX0
457         self.alpha = np.arccos((b**2 + c**2 - a**2)/(2*b*
                c))
458         self.beta = np.arccos((a**2 + c**2 - b**2)/(2*a*c
                ))
459         self.gamma = np.arccos((a**2 + b**2 - c**2)/(2*a*
                b))
460
461         IQ0,IQ1 = get_data()
462         Data = np.array([IQ0,IQ1])
463         try:
464             Calibrate(Data)
465         except:
466             print("Impossible to calibrate")
467
468     def onMyToolBarButtonClick4(self, s):
469         #Button 4 : Stops (in a clean way) the
                acquisition
470         print("click", s)
471         end_acquisition(UDP0,UDP1)
472
473     def onMyToolBarButtonClick5(self, s):
474         #Button 5 : Clean the UDP communication
475         print("click", s)
476         clean_udp(UDP0,UDP1)
477
478     def onMyToolBarButtonClick6(self, s):
479         #Button 6 : Turns ON the motor of ESP300
480         print("click", s)
481         self.ESP300 = MotorOn()
482         print("ESP300 motor is ON")
483
```

```
484     def onMyToolBarButtonClick7(self, s):
485         #Button 7 : Resets the position and turns OFF the
                motor of ESP300
486         print("click", s)
487         self.Box6.setValue(0)
488         #self.Box7.setValue(0)
489         MotorOff(self.ESP300)
490         print("ESP300 motor is OFF")
491 
492     def onMyToolBarButtonClick8(self, s):
493         #Button 8 : Gets and displays the (applied) phase
                shift
494         print("click", s)
495         self.lcd1.display(self.k*self.delta_dist)
496 
497     def onMyToolBarButtonClick9(self, s):
498         #Button 9 : Gets and displays the current X
                position of ESP300
499         print("click", s)
500         X = GetX(self.ESP300)
501         self.lcd1.display(X)
502 
503     def onMyToolBarButtonClick10(self, s):
504         #Button 10 : Gets and displays the (applied)
                phase shift
505         print("click", s)
506         Y = GetY(self.ESP300)
507         self.lcd1.display(Y)
508 
509     def onMyToolBarButtonClick11(self, s):
510         #Button 11 : Gets and displays the (applied)
                phase shift
511         print("click", s)
512         MoveTo(self.ESP300, self.ESP300_X, self.ESP300_Y)
513         print("Motion has ended")
514 
515     def onMyToolBarButtonClick12(self, s):
516         #Button 12 : Requests the flag
517         print("click", s)
518         flags = request_flag(UDP0, UDP1, self.flag_idx)
```

```
519
520         if (self.flag_idx == 0):
521             self.lcd1.display(flags[-1])
522         elif (self.flag_idx == 1):
523             self.lcd1.display(flags[-5])
524
525     def onMyToolBarButtonClick13(self, s):
526         #Button 13 : Reset the flags
527         print("click", s)
528         reset_flag(UDP0, UDP1)
529
530
531
532     def update_plot(self):
533
534         IQ0,IQ1 = get_data()
535         #lambd = 3e8/434e6
536         #k = 2*np.pi/lambd
537
538         if (self.UpdatePhaseShiftFlag):
539             #IQ1 = IQ1*np.exp(-1j*self.k*self.delta_dist)
540             phi_factor = 2.472e9/434e6
541             IQ0 = IQ0*np.exp(-1j*self.k*self.delta_dist*
                    phi_factor)
542         Data = np.array([IQ0,IQ1])
543         UpdatePowFlag = 0
544         #Try to process the data.
545         #If it fails (because of invalid data), raises an
                exception and sets the flag to 0
546         try:
547             powdb = processing(Data,self.posx)
548             UpdatePowFlag = 1
549         except:
550             print("No data to process")
551             UpdatePowFlag = 0
552
553         #Update the data for the plots
554         self.ydata10 = IQ0.real
555         self.ydata11 = IQ1.real
556         self.ydata20 = IQ0.imag
```

```
557             self.ydata21 = IQ1.imag
558             self.ydata31 = self.DoA
559 
560             #Particular case for MUSIC power
561             if UpdatePowFlag :
562                 self.ydata30 = powdb
563             else:
564                 #If no data to process plot a random thing
565                 #self.ydata30 = self.ydata30[1:] + [random.
                        randint(0, 10)]
566                 self.ydata30 = [random.randint(0, 10) for i
                        in range(180)]
567 
568             #Update the plots by first checking if a
                    reference does exist
569             if (None in self._plot_ref):
570                 plot_refs_1 = self.plot1.axes.plot(self.
                        xdata1, self.ydata10, 'b',label="RX0")
571                 self._plot_ref[0] = plot_refs_1[0]
572                 plot_refs_2 = self.plot1.axes.plot(self.
                        xdata1, self.ydata11, 'r', label="RX1")
573                 self._plot_ref[1] = plot_refs_2[0]
574 
575                 self.plot1.axes.legend()
576                 self.plot1.axes.set_ylim(-2000,2000)
577 
578                 plot_refs_3 = self.plot2.axes.plot(self.
                        xdata2, self.ydata20, 'b', label="RX0")
579                 self._plot_ref[2] = plot_refs_3[0]
580 
581                 plot_refs_4 = self.plot2.axes.plot(self.
                        xdata2, self.ydata21, 'r', label="RX1")
582                 self._plot_ref[3] = plot_refs_4[0]
583 
584                 self.plot2.axes.legend()
585                 self.plot2.axes.set_ylim(-2000,2000)
586 
587                 plot_refs_5 = self.plot3.axes.plot(self.
                        xdata3, self.ydata30, 'b', label = "MUSIC"
                        )
```

```
588                 self._plot_ref[4] = plot_refs_5[0]
589
590                 plot_refs_6 = self.plot3.axes.plot((self.
                        ydata31,self.ydata31),(self.xdata30,self.
                        xdata31),scaley=False, color = "green",
                        linestyle='--', label = "DoA")
591                 self._plot_ref[5] = plot_refs_6[0]
592
593                 self.plot3.axes.legend(loc="upper right")
594
595             else:
596                 # We have a reference, we can use it to
                        update the data for that line.
597                 self._plot_ref[0].set_ydata(self.ydata10)
598                 self._plot_ref[1].set_ydata(self.ydata11)
599                 self._plot_ref[2].set_ydata(self.ydata20)
600                 self._plot_ref[3].set_ydata(self.ydata21)
601                 self._plot_ref[4].set_ydata(self.ydata30)
602                 self._plot_ref[5].set_xdata(self.ydata31)
603
604             # Trigger the canvas to update and redraw.
605             self.plot1.draw()
606             self.plot2.draw()
607             self.plot3.draw()
```

Listing 10: Processing.py

```
1  import numpy as np
2  import sys
3  sys.path.append('../PostProcessing')
4  from CorrMat import *
5  from MUSIC import *
6
7  def processing(data,posx):
8      cal = 0
9      posy = np.array([0,0])
10     lambd = 3e8/2.472e9
11     cor = CorrMat(data, cal)
12     powdb = MUSIC(cor, posx, posy, lambd)
13
14     return powdb
```

```
15
16 def Calibrate(data):
17     cor = CorrMat(data, 1)
18     return
19
20 def get_angle(x):
21     d = 1.635
22     Ang = 90 - np.rad2deg(np.arccos(d/np.sqrt(d**2+x**2))
           )
23     #Ang = 90 - np.arctan(x/d)
24     return Ang
```

Listing 11: ESP300.py

```
1 import pyvisa
2 import time
3
4 def MotorOn():
5     rm = pyvisa.ResourceManager()
6     #Open Instrument Communication
7     print('open ESP300 Communication')
8     ESP300 = rm.open_resource('GPIB0::1::INSTR')
9
10    print(ESP300.query('*IDN?'))
11
12    #Motors ON
13    print('Turn ON motors')
14    ESP300.write('1MO\n')
15    #ESP300.write('2MO\n')
16    print(ESP300.query('1VA?'))
17
18    time.sleep(1)
19    return ESP300
20
21 def MotorOff(ESP300):
22    #Reset Position
23    print('Return to 0,0')
24    ESP300.write('1PA 0;1WS100\n')
25
26    #*ESP300.write('2PA 0;2WS100\n')
27
```

```
28      #time.sleep(2);
29
30      #Turn OFf Motors
31      print('Turn␣OFF␣motors')
32      ESP300.write('1MF\n')
33      #ESP300.write('2MF\n')
34
35      #Close Instrument
36      print('Close␣Instrument')
37      ESP300.close()
38      return
39
40  def SetX(ESP300,Target_X):
41      print('Go␣to␣X␣=␣' + str(Target_X))
42      ESP300.write('1PA' + str(Target_X) + ';1WS100\n')
43      return
44
45  def SetY(ESP300,Target_Y):
46      print('Go␣to␣Y␣=␣' + str(Target_Y))
47      ESP300.write('2PA' + str(Target_Y) + ';2WS100\n')
48      return
49
50  def MoveTo(ESP300,Target_X,Target_Y):
51      print('Goto␣' + str(Target_X) + ',' + str(Target_Y))
52      ESP300.write('1VA4')
53      print(ESP300.query('1VA?'))
54      ESP300.write('1PA' + str(Target_X) + ';1WS100\n')
55      #ESP300.write('1PA' + str(Target_X))
56      #ESP300.write('2PA' + str(Target_Y) + ';2WS100\n')
57      """
58      Current_X = 1000.0
59      while Current_X != Target_X:
60          Current_X = float(ESP300.query('1TP'))
61          print(Current_X)
62          time.sleep(1)
63      #print("Motion has ended")
64      """
65      return
66
67  def GetX(ESP300):
```

```
68      Current_X = float(ESP300.query('1TP'))
69      return Current_X
70
71  def GetY(ESP300):
72      Current_Y = float(ESP300.query('2TP'))
73      return Current_Y
```

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**