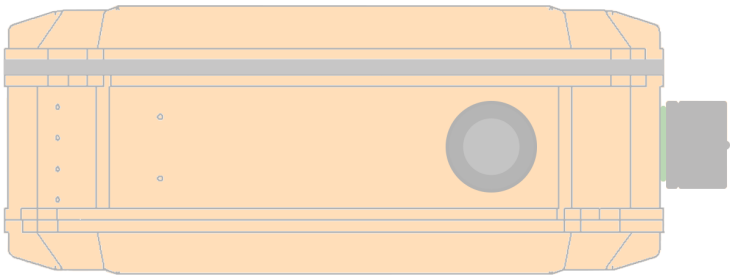
UAB

Master's Thesis

Master in Telecommunication Engineering

Experimentation with SDR sensors for cloud GNSS signal processing

Joaquín Gáñez Fernández



Thesis Advisor: Jose A. López Salcedo

Department of Telecommunications and Systems Engineering

Escola d'Enginyeria (EE)

Universitat Autònoma de Barcelona

Bellaterra, February 2021

El sotasignant, José A. López-Salcedo, Professor de l'Escola d'Enginyeria (EE) de la Universitat Autònoma de Barcelona (UAB).

CERTIFICA:

Que el projecte presentat en aquesta memòria de Treball Fi de Grau ha estat realitzat sota la seva direcció per l'alumne Joaquim Gáñez Fernández.

I, perquè consti a tots els efectes, signa el present certificat.
Bellaterra, 15 de Febrer de 2021.

Signatura:

Resumen: La implementación de técnicas de posicionamiento GNSS convencinales en dispositivos IoT, en los que el bajo consumo energético suele ser siempre una prioridad, no es viable en la mayoría de los casos debido a que éstas requieren un consumo energético demasiado elevado. Gracias a los avances tecnológicos de los últimos años, se han desarrollado nuevas técnicas que consiguen liberar al dispositivo de la gran mayoría del trabajo, reduciendo drásticamente su consumo de energía además de aportar muchas otras ventajas. De acuerdo con esta premisa y bajo la atmósfera del proyecto europeo "Navigation and GNSS in Smart Cities – Testbed Concept Definition (HANSEL)", el objetivo del esutdiante será diseñar, desarrollar y validar un prototipo de sensor capaz de capturar, almacenar y transmitir muestras de señal GNSS y de redes celulares para su posterior procesado remoto en un servidor externo. Para lograrlo, el estudiante deberá aprender sobre, entre otras tecnologías, posicionamiento GNSS mediante el procesado de fragmentos de señal, posicionamiento GNSS asistido y captura de señales de radiofrecuencia mediante el uso de dispositivos basados en Software Defined Radio. También deberá desarrollar su autonomía y creatividad para buscar la forma de superar los retos que cada tarea presenta.

Resum: La implementació de tècniques de posicionament GNSS convencinales en dispositius IoT, en els quals el baix consum energètic sol ser sempre una prioritat, no és viable en la majoria dels casos a causa de que aquestes requereixen un consum energètic massa elevat. Gràcies als avenços tecnològics dels últims anys, s'han desenvolupat noves tècniques que aconsegueixen alliberar el dispositiu de la gran majoria de la feina, reduint dràsticament el seu consum d'energia a més de moltes altres avantatges. D'acord amb aquesta premissa i sota l'atmosfera del projecte europeu "Navigation and GNSS in Smart Cites - Testbed Concept Definition (HANSEL)", l'objectiu de l'estudiant serà dissenyar, desenvolupar i validar un prototip de sensor capaç de capturar, emmagatzemar i de transmetre mostres de senyal GNSS i de xarxes cel · lulars per al seu posterior processament remot en un servidor extern. Per aconseguir-ho,l'estudiant haurà d'aprendre sobre, entre d'altres tecnologies, posicionament GNSS mitjançant el processament de fragments de senyal, posicionament GNSS assistit i captura de senyals de radiofreqüència mitjançant l'ús de dispositius basats en Software Defined Radio. També haurà de desenvolupar la seva autonomia i creativitat per buscar la manera de superar els reptes que cada tasca presenta.

Summary: The implementation of conventional GNSS positioning techniques in IoT devices, in which low energy consumption is always a priority, is not feasible in most cases because they require too much energy consumption. Thanks to the technological advances of recent years, new techniques have been developed that manage to free the device from the vast majority of work, drastically reducing its energy consumption in addition to many other advantages. With this premise and under the atmosphere of the European project "Navigation and GNSS in Smart Cities - Testbed Concept Definition (HANSEL)", the objective of the student will be to design, develop and validate a prototype of sensor capable of capturing, storing and transmitting GNSS and Cellular signal fragments for later remote processing on an external server. For this, the student must learn about technologies such as GNSS positioning by processing signal fragments, assisted GNSS positioning, and capturing radio-frequency signals through the use of devices based on Software Defined Radio, among others. He will also need to develop his autonomy and creativity to find a way to overcome the challenges that each task presents.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Internet of Things (IoT) has started changing the world. Everyday more and more connected devices emerge in every aspect of our live generating functionalities that would have seemed impossible few years ago. For some applications, the knowledge of devices position accurately is a must as well as the low power consumption. While positioning by terrestrial networks offer a low-power consumption positioning service, its performance in terms of accuracy is not even close to the Global Navigation Satellites Systems (GNSS) positioning but the power consumption GNSS positioning use to require its far away to be feasible for most IoT applications.
Most of the power consumption in GNSS positioning devices is caused by the complex computations required to obtain the position estimations. Cloud computing together with current networks may be a game changer for this problematic since with the appropriate infrastructure and witht the use of state-of-the-art techniques, those complex computations can be carried out by an external server, hence relieving the sensor from the hard work.

Under this context, the UAB contribution to the European Space Agency founded project "Navigation and GNSS in Smart Cities – Testbed Concept Definition" (codenamed HANSEL), is born. Its main objective is the development of a navigation-based services platform with big data capabilities focused in Smart cities needs, where IoT plays a powerful role.

1.1 Motivation

The fact that in a near future there may be billions of interconnected devices sensing and gathering data for its later storage and processing is very interesting. "Data is the new oil" is a widely known phrase that indicates that having big quantities of data in an ordered structure opens a wide range of possibilities so powerful that these days it sounds innovative and trends

show that tomorrow it will be everywhere, just as it happened with websites.

Although since there is a list of requirements set, in this project, there is a wide freedom to let the creativity flow to find a way, not only to fulfill them, but to improve them as much as possible. Thus, a lot of novel GNSS positioning and signal gathering techniques research will be made in order to contribute in technological advances as much as possible.

1.2 Objectives

The main objective of the whole HANSEL project is to develop a testbed to demonstrate a series of services around navigation and localization that could be implemented in the context of Smart Cities. Such services are based on GNSS technologies as well as wireless communication signals of opportunity (3G/4G/5G, WiFi).

Regarding the scope of this thesis, the main objectives are the design, develop and validation of a set of low cost connected devices (sensors) capable of remotely gathering signal for its later external processing and storage, which includes:

- Selection and testing of hardware components.
- Development of the sensor software.
- Development of the necessary software to remotely orchestrate the sensors.
- Design of validation tests.
- Development of the necessary software to process and visualize testing results.
- Understand the relevance of signal gathering and processing parameters by means of experimentation.

1.3 Methodology

Before starting to performing development tasks, the gathering of the necessary knowledge of project related topics and the project itself had to be performed. This included the extensive reading of GNSS related articles as well as technical documents, papers, the HANSEL project proposal and statement of work and many more.

As the work performed was very diverse, it was divided by tasks. A task could either be the integration of a new hardware component in the sensor, the rework of the sensor design or the development of a software package.
The completion of every task followed, more or less, the following structure:

1. Research: (If needed) this could go from the reading of an SDR manual (when available) to the communication interface specifications, but almost in every task some research had to be made.

2. Concept design: High level design of the future implementation, every requirement and dependency must be taken into account.

3. Concept implementation: Once concept design is validated, the implementation process is performed. Here is where the relevance of the previous point is reflected.

4. Testing phase: Once the implementation is performed, every functionality must be tested in order to be validated. If any functionality does not pass the testing phase, go back to step 2. If testing phase is declared passed, a new task can be started.

1.4 Document structure

Chapter 2 explains the framework of the project and makes an introduction to some theoretical concepts whose understanding is recommended in order to appreciate the work.

Chapter 3 explains the development stage of the sensor prototypes. It starts with the concept design (3.1) and the hardware components selection (3.2). After that, the final prototype design is shown and explained (3.3). Furthermore, the developed software package the sensor contains is defined in section (3.4) and the full user-sensor interaction process is explained in section (3.5)

Chapter 4 explains the sensor validation phase. Since some software tools were developed to process and visualize test results, they are defined in section (4.1) and after that the validation process for the different sensors is presented in section (4.2).

Chapter 5 contains the three most relevant experimentation tests that were made with the SDR sensors using the cloud GNSS receiver developed by the SPCOMNAV/UAB group as an external service that processed the samples gathered by the sensors developed herein.

Chapter 6 contains the conclusions extracted from this experience. Section 6.1 describes a published article which is based on the results obtained in this thesis.

Chapter 2

The Hansel positioning testbed and other concepts

2.1 Context of the dissertation: The HANSEL project

The purpose of this report is to capture the experience and all the knowledge gained during the materialization process of the master's degree dissertation, which was performed in the research group SPCOMNAV. Every aspect regarding the work represented in this document is framed in the European Space Agency founded project "Navigation and GNSS in Smart Cities", code-named as "HANSEL" and from this point of the report this is how it will be referred as.

HANSEL is articulated by a set of services that are encapsulated within Docker containers. Each Docker service interacts with other components (e.g. other services or external applications) by means of an API that provides a homogeneous and standardized access point, the mentioned structure is depicted in Figure 2.1.

As it can be observed in Figure 2.1, HANSEL is composed by a variety of positioning related services. In total, four companies and institutions work together in this project:

- Rokubun (ROK): An IT company located in Barcelona.
- Traffic Now (TFN): An IT company located in Barcelona.
- Politecnico di Torino (PTT): A university located in Torino, Itally.
- SPCOMNAV (UAB): A research group from Universitat Autònoma de Barcelona. Located in Cerdanyola, Barcelona.

The SPCOMNAV contribution consists in two parts: SNAP which stands for "Snapshot

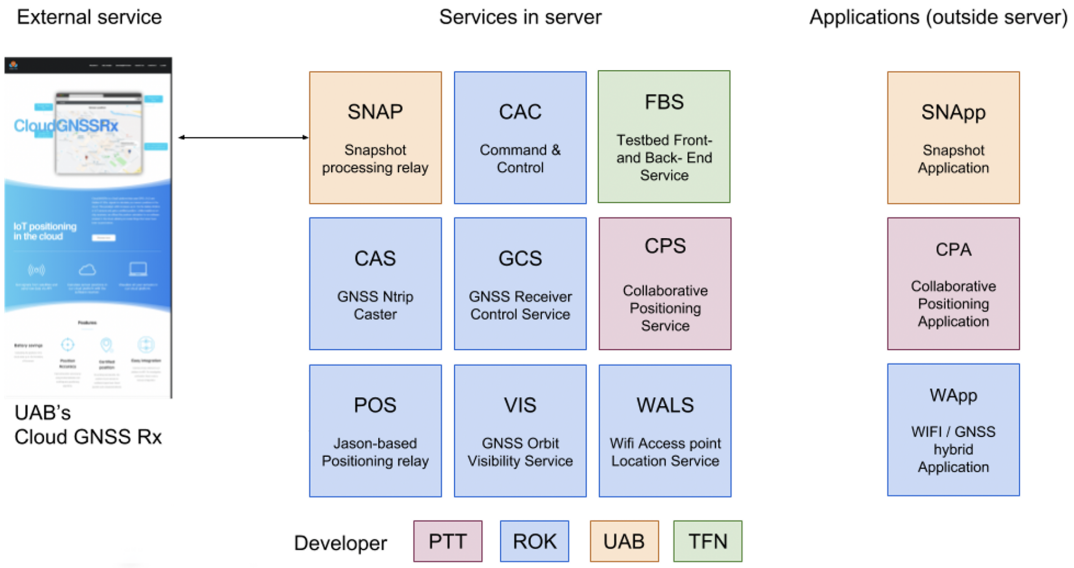


Figure 2.1: HANSEL project structure.

processing relay" and SNApp which stands for SNAP Application. Together, SNAP and SNApp constitute a remote processing platform of GNSS and Cellular (LTE and 5G) signals that brings, among other services, the possibility of obtaining position estimations by means of raw signal fragments for the latter visualization on a web-based Big Data service called Front-and-Back-End Service (FBS).

The SNAP service is built over three different services: SNAP-G, SNAP-C, and SNAP-H. SNAP-G and SNAP-C provide the testbed with the user's or sensor position and observables using GNSS and cellular infrastructures, respectively. The SNAP-H service provides the hybrid position using the GNSS and cellular observables given by the SNAP-G and SNAP-C services, respectively. A high-level perspective of the SNAP service is depicted in figure 2.2 and its main components are listed below.

- SNAP DB, a SQL database based in PostgreSQL which stores all the configurations and position fixes from SNAP sensors, either cellular or GNSS-based.
- SNAP API, the entry point of the service that interfaces the external world to the subservices SNAP-G, SNAP-C and SNAP-H as well as the subservices with SNAP DB. The purpose of this components is also to validate the input parameters coming from the user and report if there is any error.
- SNAP G, a subservice whose main function is to relay GNSS raw measurements to the external service CloudGNSSrx for its processing, for the later introduction of the results

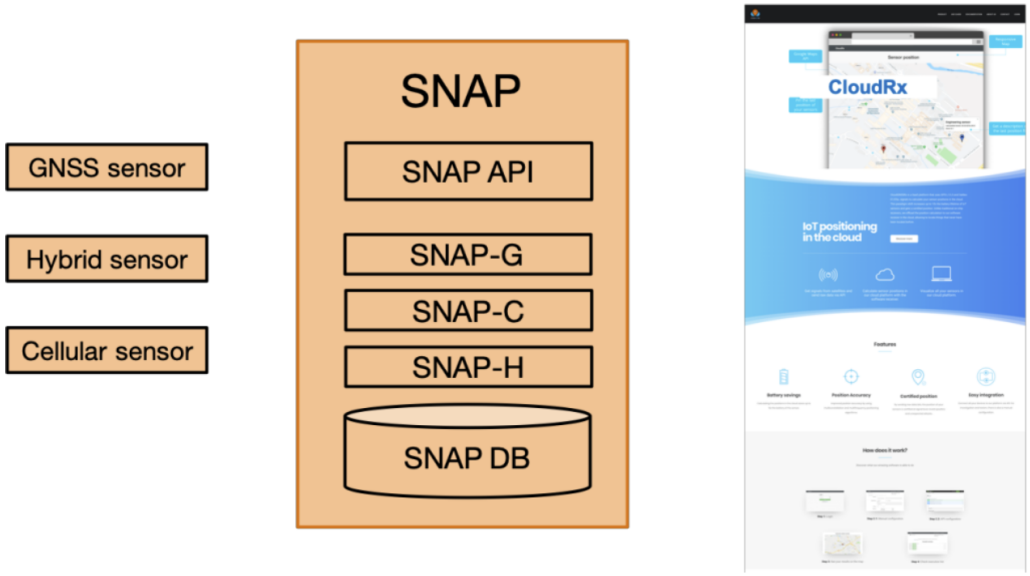


Figure 2.2: SNAP service components.

in SNAP DB.

- SNAP C, a subservice whose mainfunctions are a) to relay LTE real samples to the external service in order to its processing, and b) to simulate cellular-based position fixes. All the related results are later introduced in SNAP DB.
- SNAP H, a subservice whose main function is to hybridize GNSS observables and cellular observables in order to obtain a hybrid position fix (which is later introduced in SNAP DB)

Following the same nomenclature, the different sensors are listed below:

- GNSS or G sensors: Sensors capable of gathering GNSS signals by means of a radiofrequency front-end and sending them via Internet to the service for its external processing.
- Cellular Physic or C sensors: These sensors gather Cellular signal by means of a radiofrequency front-end, with the main objective of sending them to the external service and computing the raw Cellular observables (not suitable for position computation).
- Cellular Logic or CL sensors: These sensors are not a physical entity, they are software based. They are in charge of triggering a simulation in order to obtain corrected Cellular observables and compute a simulated Cellular position with them. They were created to simulate a real-world scenario, where it would be possible to obtain the position out of real Cellular signal if there were no constraints. In this document this sensors are not present.

- Hybrid or H sensors: These sensors are in able to both capture real GNSS signals and trigger the simulation of corrected Cellular observables, with the main objective of measurement hybridization. These sensors are needed since the hybridized measurements necessarily need to have the same reference position, so instead of having two sensors (one G and another CL) in the same position, it was decided to create the hybrid sensors for simplicity reasons. That means if there were no limitations computing cellular positions out of real signal, this type of sensor would be a physical sensor with both GNSS and Cellular front-ends. From the physical sensor operation side, these sensors operation are exactly the same as GNSS sensors.

In figure 2.3 the interaction of the sensors with SNAP services is depicted, as well as its the interaction of snap and the external service testbed Front-and-Back-End service (FBS) which consists in a web-based platform whose main functionalites are, among others, to manage SNAP service and visualize SNAP DB data.

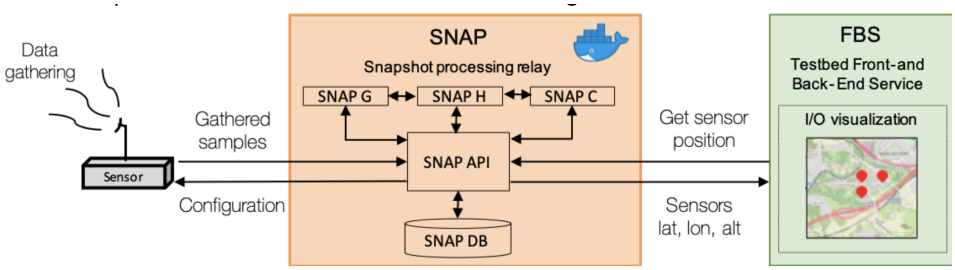


Figure 2.3: SNAP service component interactions.

SNApp is composed by all the logic the SNAP sensors contain, which make them able to capture and transmit signal fragments in a very flexible and fully autonomous way. SNApp is described in greater depth in section 3.4.

2.1.1 The CloudRx service

The CloudRx, also called CloudGNSSRx, is the external service that complements the SNAP service of the HANSEL Testbed, powered by Amazon Web Services and developed by the SP-COMNAV research group at UAB. The CloudRx service is used in the scope of this thesis as an external service that processes the samples gathered by the developed sensors, the latter being the core of the present work. Users communicating with the Testbed server will have access to the cloud GNSS receiver where snapshot processing and cooperative localization of jammers will take place. The basic principle of the platform is the externalization of the signal processing and position computation to a cloud platform, contrary to the (classical) procedure of the actual receivers, performing the position computation in situ. This is performed by means of sending to the cloud the gathered raw measurements for them to be processed.

2.1.2 SNAP-G GNSS snapshot relay service

SNAP-G provides an interface for processing snapshots of GNSS raw samples data gathered by GNSS sensors. This service is be implemented as a relay to UAB's cloudGNSSrx service (http://cloudGNSSrx.com): the cloudGNSSrx API is used by the Testbed in order to upload the required files, wait until the process is completed, and then display the solution. A simplified block diagram of the SNAP-G relay service is included in the figure 2.4. The GNSS sensor includes a JSON with a predetermined configuration, which can be displayed through the FBS. After detecting a change (e.g., the user is launching a new execution on-demand), the GNSS sensor capture a snapshot of the GNSS signals using the configuration parameters and forward it to the SNAP-G relay service, which relays its input to the UAB's cloudGNSSrx. After a short processing time, the cloudGNSSrx returns to the SNAP-G the PVT of the sensor, the status of the GNSS signals, the generated observables (i.e., pseudoranges), and, if this option is selected, the location of any interference source that may be detected in the surroundings of the GNSS sensor. The output of the cloudGNSSrx is be stored in a database of the SNAP-G relay service, which in the end can be displayed in the FBS. SNAP-G requirements are listed in table 2.1.

Ref	Description
REQ-SNAPG-01	The service SHALL provide the position of a GNSS sensor based on the input GNSS raw samples file.
REQ-SNAPG-03	The service SHALL gather, decode and process GNSS raw samples files with different formats for future SDR additions within the Testbed.
REQ-SNAPG-04	The service SHALL modify the configuration parameters of the external device and cloudGNSSrx for different GNSS processing approaches.

Table 2.1: SNAP-G service requirements.

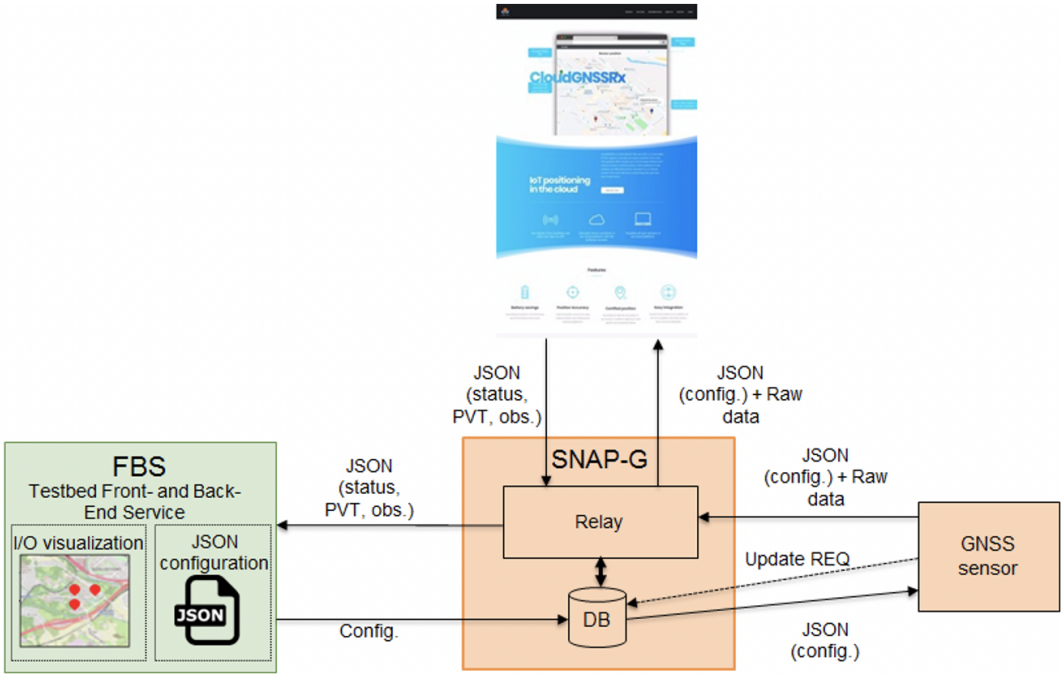


Figure 2.4: SNAP-G service block diagram.

2.1.3 SNAP-C Cellular snapshot relay service

SNAP-C provides an interface to obtain observables from real cellular signals (4G, extensible to 5G). The service also allows the option to internally generate cellular observables and PVT measurements from simulated 3G/4G/5G cellular signals.

A simplified block diagram of the SNAP-C service is included in 2.5. The main goal of this service is to provide cellular observables and positions. The SNAP-C service is split into two different components:

- A relay service in charge of forwarding real 4G signals that may eventually be gathered by cellular sensors to the UAB's cloudGNSSrx, where they are be processed. It is important to note that the cloudGNSSrx allow the processing of either GNSS or cellular measurements. Once the observables have been processed at the cloudGNSSrx, they are forwarded back to the SNAP-C where are stored on its own database and displayed in the FBS. The cellular sensors that make use of the SNAP-C relay service are the so-called physical ones or "C/Cellular sensors", since they are hardware sensors able of gathering real signals according to some user configurable parameters. The SNAP-C relay functionality is depicted in the upper part of 2.5, inside SNAP-C module.
- A software that simulates a 3G/4G/5G cellular infrastructure and provides observables

and/or position solutions. The user shall configure the simulation parameters (e.g., reference position, BS location, positioning method, etc.) through the FBS. Then, a deployment of base stations is performed given the input configuration parameters and the working scenario. This information will be used to generate simulated cellular observables, which finally will be used to compute the PVT. The results will be stored in the SNAP-C database along and will be available and depicted at the FBS. The cellular sensors that make use of this simulation module are the so-called logical sensors, since they are actually virtual sensors part of the software simulation in charge of generating cellular observables and PVT. The components of this module can be seen in the lower part of Figure-UAB 1, inside SNAP-C service. Note that logical sensors appearing with a dashed line in Figure-UAB 1 are actually contained within the SNAP-C. They have been represented as an external block in Figure-UAB 1 just for illustration purposes, to make the functional parallelism with the physical (i.e. hardware) sensors. This component is out of the scope of this document.

The requirements for this service which are in the scope of this document, are listed in the table 2.2

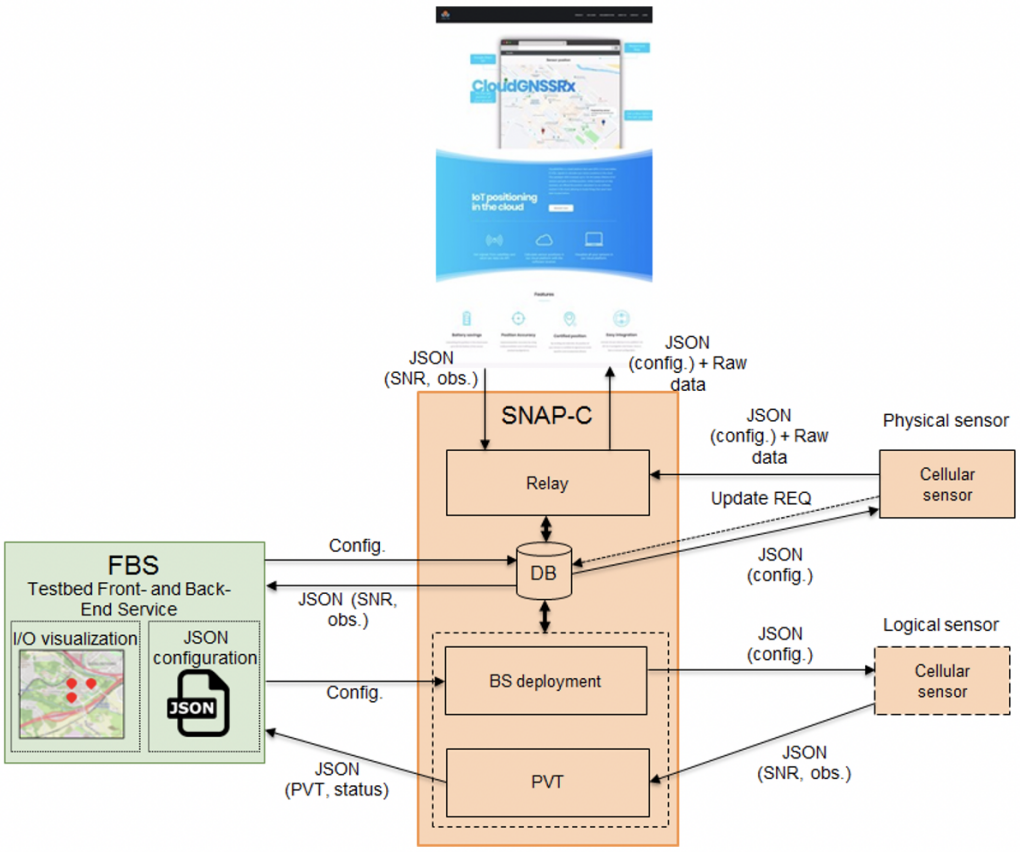


Figure 2.5: SNAP-C service block diagram.

Ref	Description
REQ-SNAPC-01	The service SHALL provide the observables of a cellular sensor based on the input raw samples file of real 4G cellular signal.
REQ-SNAPC-05	The service SHALL offer an API by which to report the status of the Testbed sensors (e.g., external device).

Table 2.2: SNAP-C service requiremients

2.1.4 SNAP-H GNSS and cellular hybrid positioning

SNAP-H provides a service for computing a hybrid PVT solution from GNSS and cellular observables. It consists of a hybrid PVT algorithm that uses GNSS and cellular observables as an input. These observables can be retrieved from previous executions or can be generated on-demand by the user through the SNAP-G and SNAP-C services, respectively. The output of the SNAP-H service will be stored in the local database of the service, and will also be available to the FBS. A simplified block diagram of the SNAP-H service is depicted in the figure 2.6 and the requirements for this service are listed in the table 2.3.

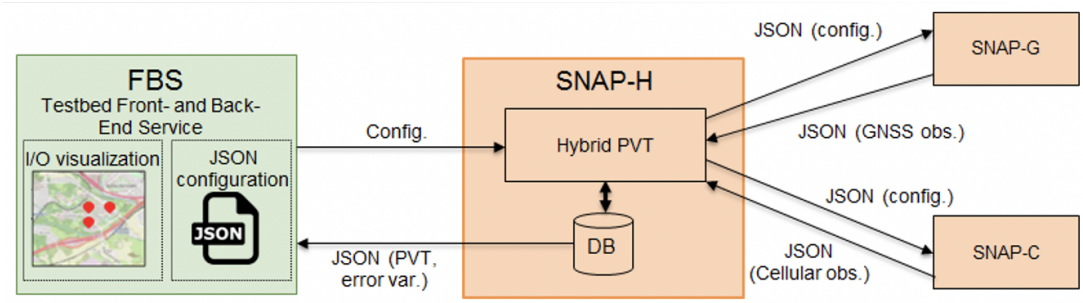


Figure 2.6: SNAP-C service block diagram.

Regarding the sensor-side, the functional point of view is exactly the same as in SNAP-G service. The only difference is that sensors using SNAP-H must be registered as ”Hybrid” sensors in the platform and not ”GNSS” sensors.

2.2 GNSS Overview: GPS and Galileo

In this section a brief description of the operation of GNSS systems GPS and Galileo are made in order to introduce the key concepts for a better understanding of the following chapters of the document.

Ref	Description
REQ-SNAPH-01	The service SHALL provide the hybrid PVT of a GNSS and/or cellular sensor based on the input GNSS and cellular observables.
REQ-SNAPH-02	The service SHALL communicate with the SNAP-G and SNAP-C service to obtain the GNSS and cellular observables on-demand as an input.
REQ-SNAPH-03	The service SHOULD use observables generated by the SNAP-G and SNAP-C services in previous executions.

Table 2.3: SNAP-H service requirements

2.2.1 Systems architecture

The architecture of current GNSS systems is differentiated into three fundamental segments or parts: the space segment, which comprises only the satellites, the control segment, which manages their operation, and the user segment, which includes the development of equipment for signal reception, that is, receivers. In figure 2.7, GPS architecture and its three segments are represented. In the following sections

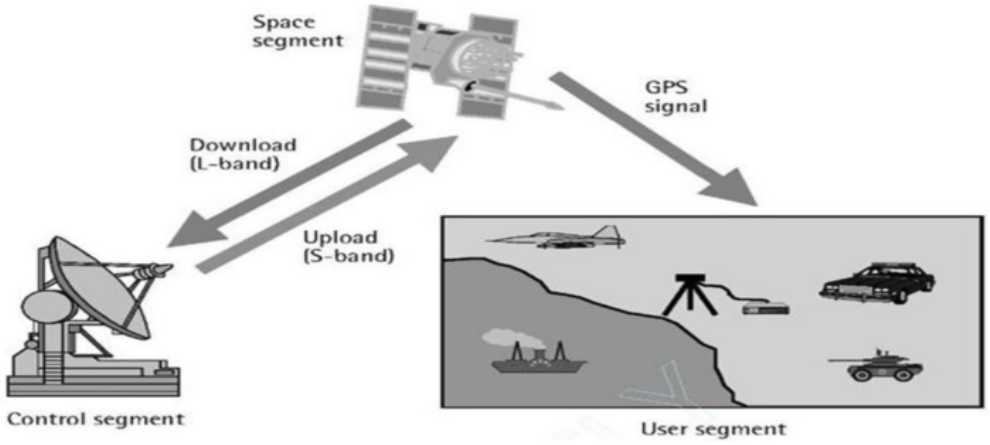


Figure 2.7: GNSS system segments.

2.2.1.1 Space segment

The space segment (SS) comprises a series of satellites (between 24 and 30, depending on the system) deployed between aproximately 19000 and 24000 kilometers of altitude (depending on the system), in the so-called intermediate circular orbit (ICO, Intermediate Circular Orbit). The orbital period, once again is dependent on the system used, is between 11 to 14 hours. The sets of satellites are located in 3 or 6 orbital planes. Due to the aforementioned layout, practically any user on Earth should be able to have a clear view (or line of sight) of 4 or more satellites

simultaneously. Satellites continuously transmit information called a navigation message, which provides any user with the data necessary to calculate their position, such as the parameters of the ephemeris, which allow the user to calculate the position of each satellite with precision, or the time correction parameters, which enable the user to calculate the clock offset and make the respective time corrections.

2.2.1.2 Control segment

The control segment (CS) is composed by a series of control base stations which communicate with the satellites which are in charge to monitor and assure the correct operation of the space segment. The user segment (US) is composed by the GNSS signal receivers at the user level, which are responsible for calculating the position. The GNSS receivers are responsible for processing the signals from the satellites with the ultimate goal of determining the user's position. In order to acquire them, users look for these signals, which travel through space, and try to synchronize with them in order to extract the relevant navigation information

2.2.1.3 Ground segment

The user segment (US) is composed by the GNSS signal receivers at the user level, which are responsible for calculating the position. The GNSS receivers are responsible for processing the signals from the satellites with the ultimate goal of determining the user's position. In order to acquire them, users look for these signals, which travel through space, and try to synchronize with them in order to extract the relevant navigation information

Despite the existence of different GNSS receiver architectures depending on the specific application, the main components are the following:

- Front end. The front end, formed by a radio frequency head or antenna. It has the main function of capturing the GNSS signals.
- Signal processing module. It has the main objective of finding GNSS signals in the signal captured by the front end. This process is very tricky as the GNSS signal power is very low (below ambient noise), so a set of complex computations is required in order to find the desired signals in the captured snapshot.
- Application module. Also known as the navigation module, it is responsible for using the data extracted in the signal processing module for various purposes, the most common being to calculate the user's position, speed and time

In the case of the architecture of the GNSS reciever studied in this project, the Front End module is placed in some autonomous devices called "SNAP Sensors" while the signal processing and application modules are placed in an external server or Cloud. This features are further defined in section 2.1.

2.2.2 Signals architecture

Broadcasting low power signals to the earth from tens of thousands of kilometers of distance is a big challenge, but it is even bigger if we take in account that every satellite in a constellation must be able to send a unique kind of signal as well as data regarding its position, clock status and so on. This gets even more complex if we take a look to the different services apart from civil positioning that GNSS constellations offer, like robust military-only signals, specific signals for high precision and dual band services and so on. In the following section, a brief summary of both GPS and Galileo E1c OS signal plan and structure is explained.

2.2.2.1 GPS Signal architecture

Actually, GPS is transmitting in three different bands: L1 band (1575.42 MHZ), L2 band (1227.60 MHz) and L5 band (1176.45MHz).

Every GPS signal has three main components:

- Carrier. The carrier is a radio frequency signal at frequency f_{L1}, f_{L2} or f_{L5}.
- Ranging code. The ranging codes correspond to a family of codes called Pseudo-Random Noise (PRN) codes. The mathematical properties of these codes allow the satellites to transmit at the same frequency without interfering with each other, due to the fact that, when these codes present the behavior of white noise, they are orthogonal to each other, so the cross-correlation is minimal, while autocorrelation is highest if the code and its replica are perfectly aligned. Codes for civil use are called Coarse/Acquisition (C/A) codes and Precise Positioning Service codes are called P(y) codes (Encrypted). Each of the GPS satellites transmits a unique C/A code and a unique P(y) code on both L1 and L2 bands. Each C/A code is composed of a sequence of 1023 bits (called chips), which are repeated in a time interval of 1ms, so the chip rate or textit chipping rate is 1.023Mcps (mega chips per second).
  The P(y) codes, as opposed to the C/A codes, are PRN codes of enormous length (around 10-14 chips), with a chip ratio of 10.23Mcps and a repetition interval of one week.
- Navigation information. The navigation message or navigation information is a binary message with various information about the sending satellite, such as its status, the pa-

rameters of the clock bias, ephemeris (position and speed) and an almanac with reduced precision information about the ephemeris of the rest constellation satellites. It takes 12.5 minutes (transmitted at 50 bps) for the entire message to be received. The essential satellite ephemeris and clock parameters are repeated each 30 seconds.

2.2.2.2 Galileo signal architecture

Galileo, plans to transmit its signals in four different bands. E1 band (1575.420 MHz), E5a band (1176.45 MHz), E5b band (1207.14 Mhz) and E6 band (1278.750 MHz). All satellites are planned to share the same frequency bands and use code-based media access methods (CDMA) similar to GPS.

Galileo signals are composed by:

- carrier. E1 OS Pilot channel is generated from the ranging and navigation message codes which are then modulated with two sub-carriers in anti-phase.
- Ranging code. E1 OS codes are 4092 chips long (4 times longer than GPS PRNs) and they are transmitted at the same rate as GPS, 1023 chips/s
- Navigation message. The Galileo satellites broadcast different types of data in four navigation messages: the F/NAV navigation, the I/NAV navigation message, the Commercial Navigation Message (C/NAV) and the Governmental Navigation Message (G/NAV).The G/NAV navigation message does not belong to the public domain and the C/NAV is not yet defined. For the case of E1 band OS, navigation message takes 10 minutes to be received. The essential satellite ephemeris and clock parameters are repeated each 30 seconds as in the case of GPS.

A difference that should be noted is that in GPS L1 C/A both ranging and navigation message codes are mixed with the the same carrier using Binary Phase Shift-Key modulation (BPSK). Hence, a 1023 Mbps code and a 50 bps code are mixed together. This causes uncertainty very 20ms $1/50s$. To avoid that, once the receiver is aligned with the navigation code bits it integrates signal fragments of 20 ms individually. On the other hand, Galileo signal is composed by two sub carriers in anti-phase keeping the PRN codes and the Navigation codes separately, this feature makes possible to process any signal length in a coherent way.

2.3 Positioning using GNSS signal snapshots

Unlike conventional techniques, snapshot techniques make the determination of position possible by using only a little fragment of a GNSS signal of some milliseconds, and therefore

it is not required to continuously capturing and processing GNSS signals. Thus, they require much less computing power and power consumption to be implemented and it makes them feasible for a wide range of IoT applications. On the other hand, devices using this techniques must have a bidirectional communication channel.
This approach allows multiple configurations which can be more or less appropriate depending on the application needs. Three possible snapshot techniques are shown below, each one has its own set of characteristics.

2.3.1 On-device Position Determination

Consists in the positioning calculations on the device. The ephemeris data is obtained externally, thus avoiding the need to extract them from the navigation message of the GNSS signal. Since recovering the ephemeris data from the navigation message takes about 30 seconds, getting it from an external source is mandatory since otherwise, more than 30 seconds of signal would be needed. Taking that into account, unlike conventional techniques, this one requires a fraction of the power and time to get a position fix, but implies the need to have access to a communications network. Although this technique avoids the need to decode the navigation message, it involves carrying out complex calculations by the sensor, which still implies high power consumption. On the other hand, the bandwidth necessary to obtain the ephemeris data is very small. This technique is graphically represented in figure 2.8.

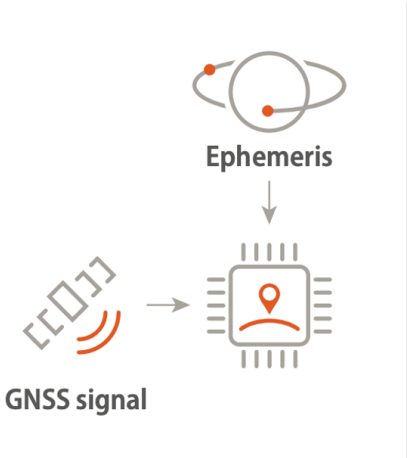


Figure 2.8: Snapshot-based on-device position determination

2.3.2 Transmission of pseudo-ranges.

This technique divides the compute load work between the device and an external server. The assistance data is obtained externally as in the previous case. The device only performs part of

the calculations obtaining the pseudoranges which are subsequently sent to the cloud where the remaining processing is finished and thus the position estimation is obtained. Although part of the calculations are performed in the cloud, the computation performed by the device is still intensive , since obtaining the pseudoranges is the most complex part. On the other hand, the bandwidth required to obtain the assistance information and send the pseudoranges data to the external server is very small. This technique is graphically represented in figure 2.9.

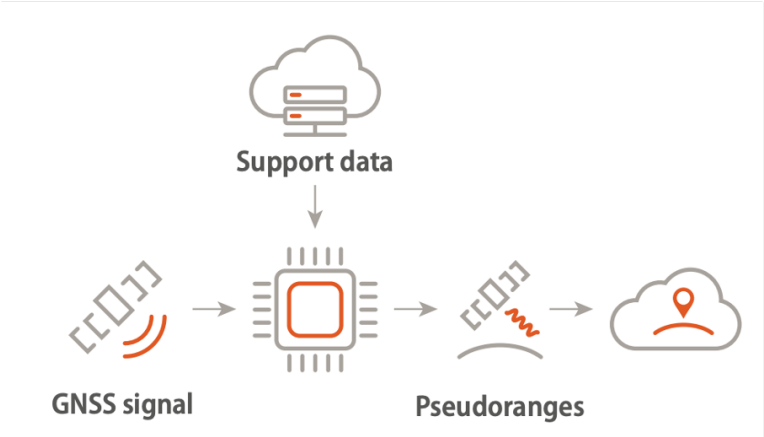


Figure 2.9: Transmission of pseudoranges obtained usign signal snapshots

2.3.3 Transmission of raw snapshots

It is based on exempting the device from any position estimation computing. To do this, the binary file of the raw captured signal fragment is sent to the external server where it is subsequently processed. This way, the power consumption of the device is drastically reduced. On the other hand, the bandwidth required for the transmission of sample files, typically a few Kb, is higher than in the previous cases. Finally, it is worth highlighting the device-processing independence provided by this technique, which allows the possibility of adding functionalities to the system without the need to make modifications to the devices, such as the implementation of interference detection or multi-constellation techniques, among others. This technique is graphically represented in figure 2.10, this approach is the one used in the SNAP services.

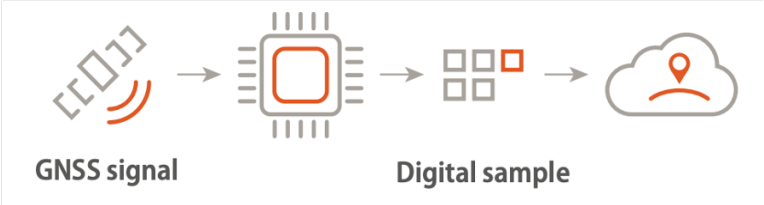


Figure 2.10: Transmission of raw snapshots

Chapter 3

Design and development of a prototype of GNSS & Cellular sensor based on Software Defined Radio technology

In recent years there has been an enormous evolution in radio-frequency (RF) devices capable of conditioning and digitizing centimeter wave signals. This is the case of the signals that are present the UHF band that extends from 300 MHz to 3 GHz., where, for example, mobile telephone services, digital television and positioning systems such as GPS or Galileo are found. The latter are also known by their acronym in English as GNSS systems (*Global Navigation Satellite Systems*). Among the different applications that have been mentioned, the case of digital television deserves a special mention since it is an open-air broadcasting service accessible to millions of people and that has a great penetration in the consumer market. It is in this particular application where RF devices have recently emerged which, thanks to economies of scale, have drastically reduced their costs [4] and made them accessible to the general public [3]. In addition, its philosophy of *Software-Defined Radio* (SDR) allows them to be flexibly reconfigured by software, making their use even more versatile in a multitude of applications [5], including instrumentation and measurement [6]. This flexibility makes it easy to implement systems using SDR devices as low cost RF heads. Although the performance is not comparable to custom designed and manufactured devices, it is sufficient for a large number of applications. The most popular ones correspond to the reception of digital television or amateur radio signals. However, the possibilities of RF devices based on the concept of SDR go much further. Specifically, this work presents the case of using low-cost RF devices to implement a signal sensor for GNSS satellite positioning systems, such as GPS or Galileo.

3.1 Sensor initial design

The RF sensor to be developed in the present thesis must fulfill some requirements:

- It shall be able to access the internet for data transmission and reception.
- It shall shall be able to gather, store and transmit GNSS and cellular signals.
- It shall be able in constant communication with the platform.
- It shall have a built-in battery lasting at least 10 hours.
- It shall have a portable design.

Considering every requirement, a first draft scheme of a sensor prototype including all necessary components was made. It is depicted in the figure 3.1.

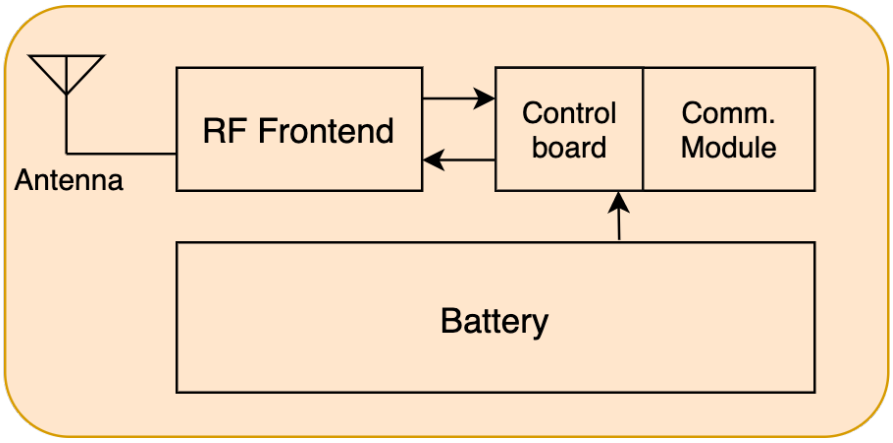


Figure 3.1: Sensor elements diagram.

The elements the sensor must contain are listed below:

- Antenna: Depending on if the sensor is a GNSS sensor (G) or Cellular (C), this element will vary between an active GNSS patch antenna or a cellular monopole.
- RF Front-end: An SDR will be used to carry out the RF samples digitization captured by the antenna. It must contain an internal bias-Tee in order to power the low noise amplifier (LNA) of active antennas for the case of GNSS sensors and it must be switchable by software.
- Control board: A control board is needed in order to control the SDR front-end. The selected board must include USB ports and a Wi-Fi module for communication purposes.

- Battery: Necessary for the prototype autonomy.
- Case: Since prototypes are intended to work outdoors, a waterproof case is preferred.

3.2 Sensor hardware components

After several research and testing, components for the device prototype were selected. The initial diagram depicted in the Figure 3.1, evolved in the one shown in the figure 3.2.

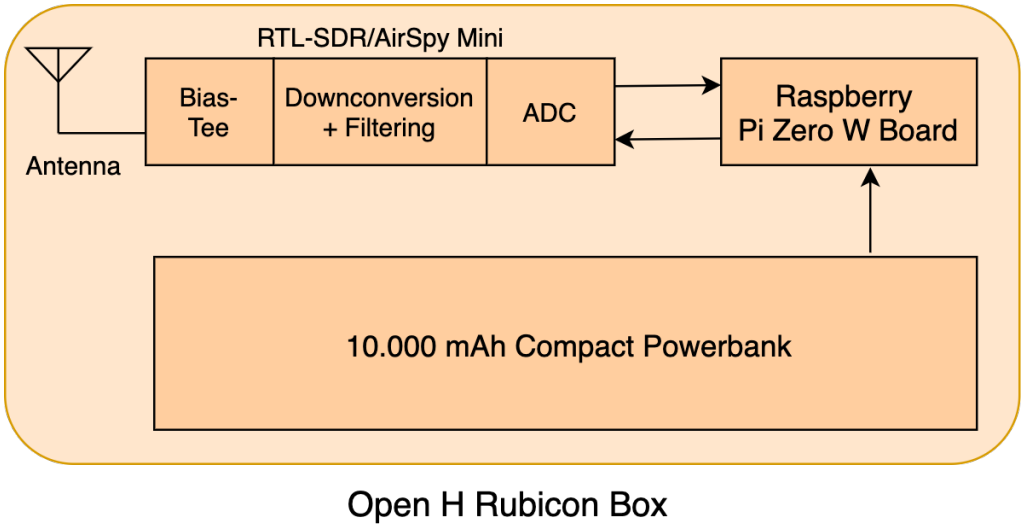


Figure 3.2: Sensor elements diagram (with selected components)

The selected components are listed below:

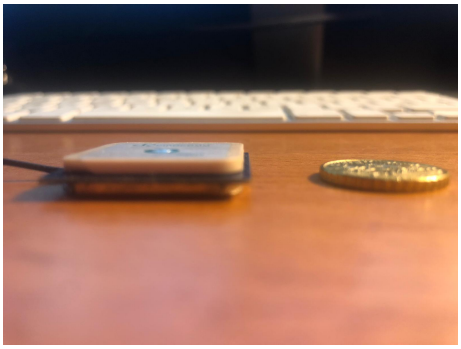
- Antenna: After testing several GNSS antennas, Circomm 580R antenna was selected due to space and connection restrictions. For Cellular sensors a generic cellular passive monopole, included in the RTL-SDRv3 developer package, will be used.
- Control Board: A raspberry pi Zero W Linux-based board with an integrated Wi-Fi module.
- RF front-end: It is attached to the control board by means of an USB port. Two SDR models are supported: RTL-SDRv3 and AirSpy Mini.
- A 10.000 mAh compact "power bank" unit. It is connected to the control board by means of an USB port.
- The Rubicon box [12] by Open H is IP67 waterproof case which is in charge of keeping all the components together in a compact and secure manner.

In the following sections, most relevant selected components are briefly described.

3.2.1 Cirocomm 580R antenna

The 580R active embedded patch antenna [11], by Cirocomm, contains a low noise amplifier which is very useful for GNSS applications since GNSS signals are received with a very low power level. What makes this model of antenna special is its dimensions which are very reduced, specially in terms of height. Its height of only 4 mm, makes it perfect for this approach taking in account the space restrictions, furthermore it uses IPEX connector where a little SMA-IPEX adapter can be placed whose dimensions and orientation once attached to the antenna IPEX port, is very convenient for this application. A couple of pictures of the antenna showing its reduced size is shown in the figure 3.3.

(a) A Cirocomm 580R antenna near a 10 euro cents coin. Top view.



(b) A Cirocomm 580R antenna near a 10 euro cents coin. Side view.

Figure 3.3: Cirocomm 580R active ceramic patch antenna.

3.2.2 Raspberry Pi Zero W board

The Raspberry Pi Zero W board (figure 3.4) is the smallest and cheapest wi-fi-integrated board of the Raspberry Pi family. The Raspberry Pi foundation aims at providing low-cost and high-performance computers for learning and leisure activities, being followed by a large community of users. In addition to its low cost and small dimensions, the Raspberry Pi Zero W board includes in-built Bluetooth and Wi-Fi antennas.

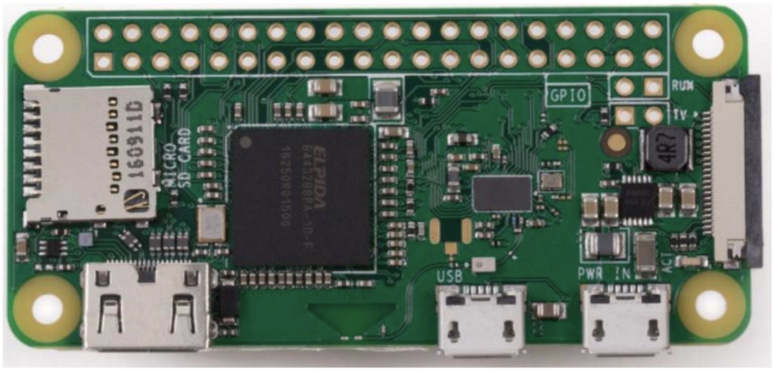


Figure 3.4: Raspberry Pi Zero W

The main features of this board are:

- Processor: single-core Broadcom BCM2835 (i.e., ARM processor) running at 1 GHz., which includes a VideoCore IV GPU.
- Memory: includes a 512MB of RAM.
- Storage: supports micro-SD cards up to 32GB for pre-built software, and large-size micro-SD cards with few format adaptations.
- Communication module: built-in wireless connectivity with 2.4 GHz. 802.11n wireless LAN, Bluetooth Classic 4.1 and Bluetooth LE technologies.
- Ports: two micro-USB ports, one for power supply (of 5V) and one for data transmission, which can be used to connect the RTL-SDR to the board with a micro-USB to USB adapter.
- Mini-HDMI: the mini-HDMI port allows to connect the board to a monitor and to easily perform the initial sensor configuration.
- Linux core: Raspbian is a Debian-based computer operating system for Raspberry Pi. This is an open-source GNU/Linux distribution, which includes Python, Java, Mathematica software among others.
- External pinout: the board has 40 GPIO (General Purpose Input/Output) ports to expand the connections of the board. This is useful to connect an external battery, such as the PiZ- UpTime Li-Ion battery board.
- Dimensions: the board has a length of 65mm, a width of 30mm and a height of 5mm.

3.2.3 RF Front-end

The RF front-end models chosen for the sensor development are the RTL-SDRv3 [13] and the Airspy Mini [14]. Both have a similar appearance, having an SMA input port for the antenna connection, and a USB output port through which digitized samples of the input signal are transmitted, and an internal Bias-Tee. Both devices are based on the concept of Software Defined Radio (SDR), and therefore are configurable. The most important features of each model are described in the following sections.

3.2.3.1 RTL-SDRv3

The RTL-SDRv3 dongle is a low-cost and configurable software defined radio (SDR) hardware. SDRs usually perform functions carried out by hardware components by means of digital processing managed by software. The RTL-SDR is composed by two main elements:

- RF front-end: based on the Rafael R820T chipset. This RF front-end works with a local oscillator that provides IF signal at the 3.57 MHz frequency.

- Digital demodulator: based on the RTL2832. The digital demodulator is fed with the IF signal, which is sampled with a sampling frequency of 2.8 MHz and a quantization of 8 bits. By means of a digital signal processor (DSP), the IF signal is demodulated and translated into a baseband signal, and hence obtaining two digital IQ samples of 8 bits at the output.

RTL-SDR includes a 1 PPM temperature compensated oscillator (TCXO). As a result, there is a clock drift that may prevent the acquisition of visible satellites for a reduced Doppler shift search range. Thus, a calibration of the RTL-SDR should be done before capturing signals in order to correct the clock drifts. In addition to the RTL-SDR, an active patch antenna is used in order to reduce the wire losses. This SDR includes a software-selectable bias-tee that provides a voltage of 4.5V at the SMA port, in order to power the LNA of the active antenna. A picture of an RTL-SDRv3 is shown in figure 3.5

Figure 3.5: RTL-SDRv3 RF front-end.

3.2.3.2 AirSpy Mini

The maximum usable and stable bandwidth of an RTL-SDR is 2.8MHz which is sufficient for GPS L1 signal capturing. Nevertheless for Galileo L1 signal captures, a higher more bandwidth is required. This is achieved with the Airspy mini RF front-end, which provides a maximum bandwidth of 6MHz that is enough for Galileo L1 signal gathering. The Airspy Mini [14] dongle is a configurable software defined radio hardware. SDRs usually perform functions carried out by hardware components by means of digital processing managed by software. The Airspy mini, similarly to the RTL-SDR, is composed by two main elements:

- RF front-end: based on the Rafael R820T2 chipset with a frequency range that extends from 24 to 1700 MHz. Its bandwidth is higher than that of the RTL-SDR reaching up to 6 MHz. The local oscillator is a TCXO with a frequency stability of 0.5 ppm
- Oversampling 12bit ADC. In Oversampling Mode, the Airspy Mini applies Analog RF and IF filtering to the signal path and increases the resolution to up to 16-bit using the software decimation.

As RTL-SDR, AirSpy mini needs to be calibrated in order to correct its internal clock drifts. It also includes a software-selectable bias-tee that provides a voltage of 4.5V at the SMA port, in order to power the LNA of the active antenna. A picture of an Airspy Mini is shown in figure 3.6.

Figure 3.6: Airspy Mini RF front-end.

The main advantage of the Airspy Mini is the higher bandwidth it offers compared to the RTL-SDR, although its price is higher (several times the price of an RTL-SDR). Therefore, it is necessary to assess the needs of the final application in order to incorporate it, in the case of this project it is needed to capture Galileo signals. In table 3.1, a comparison of the most relevant parameters of the RTL-SDR and the Airspy Mini can be observed.

	RTL-SDRv3	Airspy Mini
RF Chipset	Rafael R820T2	Rafael R820T2
Frequency range	24-1766MHz	24-1700MHz
ADC	8 bits	12 bits
Bandwidth	2.8 MHz (2.4 MHz stable)	6 MHz
Clock max. error	± 1 ppm (Theoretically)	± 0.5 ppm (Theoretically)
Internal Bias-Tee	Yes	Yes
Price	Around 20€	Around 140€

Table 3.1: RTL-SDRv3 and AirSpy Mini parameters summary.

3.2.3.3 Rubicon case

The selected case for the sensor prototype was Rubicon[12] by Open h. The main features of the case are listed below:

- High shock resistance.
- IP67 certified water resistance.
- It has a modular design composed by 4 pieces: two tapes (one transparent and one opaque

unit), a 3 mm. aluminum plate and a core module. The main advantage of this modular approach is that several combinations can be made leading to different possible designs with a single case. In figure 3.7, the 4 modules of the Rubicon case are shown.

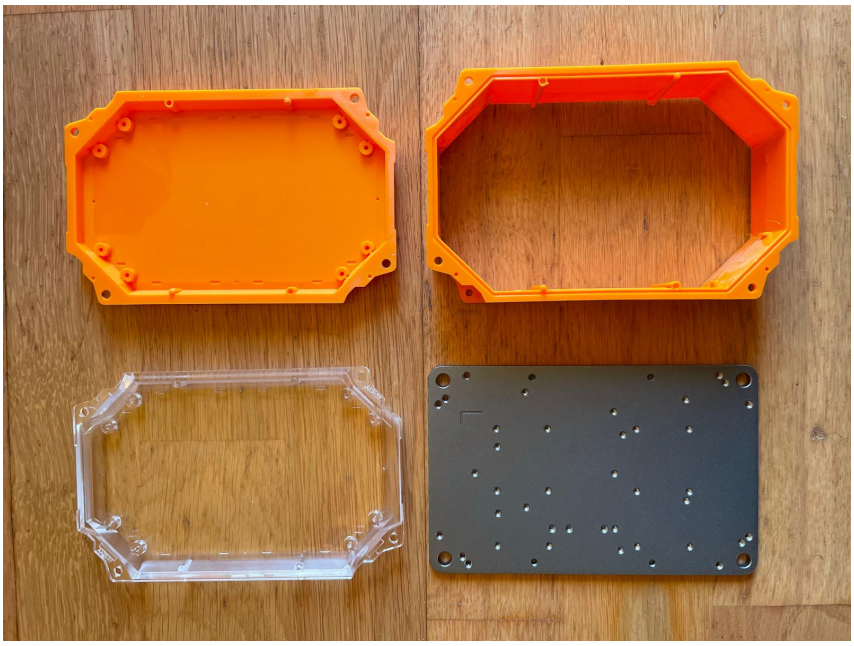


Figure 3.7: Rubicon case modules.

3.3 Sensor final prototypes

Once the hardware components were selected and the correct operation was tested and validated, a new challenge started: everything had to fit inside a case in a very tight way. During this challenge, some unexpected problems were found. In this section, first, an introduction of the final and validated assembled prototypes can be found and, after that, some unexpected challenges found during the assembly process are explained as well as the approach which was followed to surpass them.

3.3.1 GNSS/Hybrid sensor prototypes.

Two pictures of a GNSS/Hybrid sensor prototype are shown in figure 3.8. As it is mentioned in previous sections, it can be seen that the sensor case is composed by four modulesm its distribution for the GNSS/Hybrid sensors is: base tape, which is transparent so an operator can check the remaining charge in the sensors battery, a core module, a 3 mm. aluminum plate and the top tape which is opaque for light-caused heat rejection purposes. All modules are kept together by means of several removable screws. Figure 3.8(a) shows the side view of a

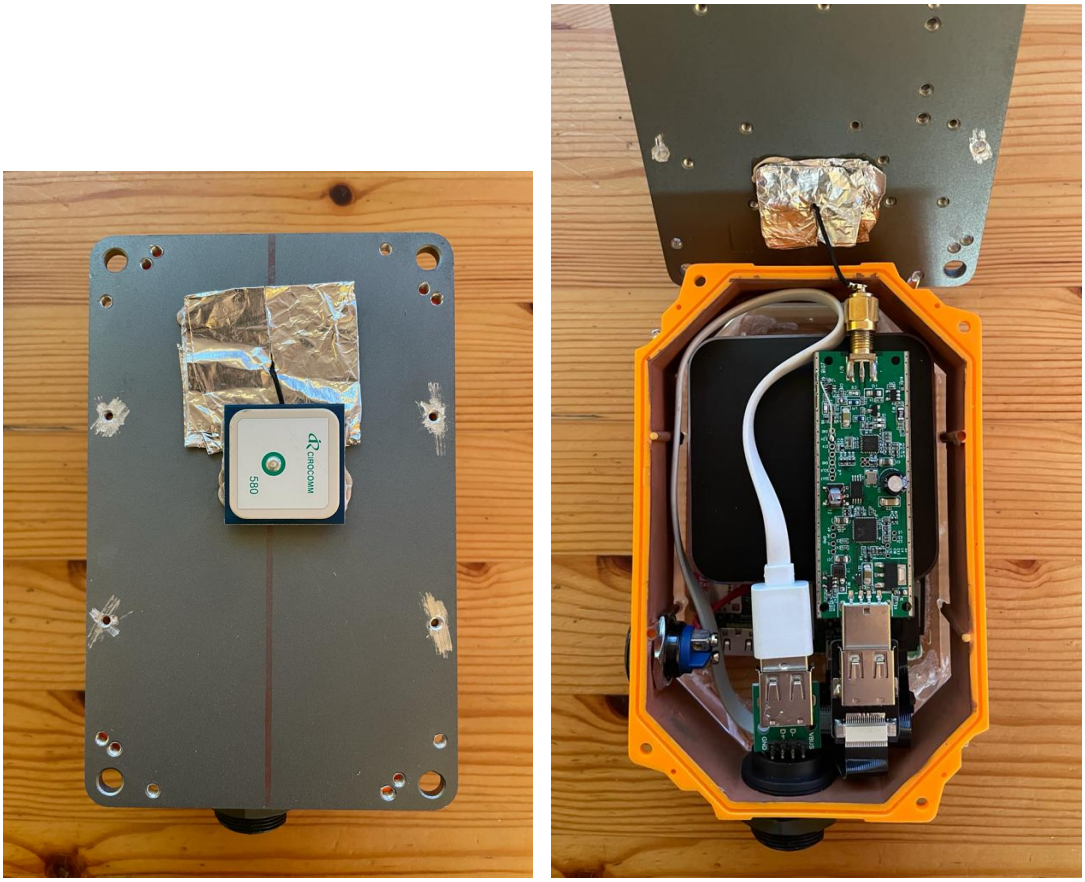
GNSS prototype, button can be distinguished from the rest whose functionality is to bring the possibility to hardware-reset the device, it also can be used as a ”power on” button. This button is composed by a simple push button which has been directly soldered to two reset dedicated pins of the Raspberry Pi Zero W. In figure 3.8(b) an USB port attached to the sensor can be observed, this port brings the possibility to charge the sensor battery without opening the case, the other side of the port is composed by 4 pins which have been manually soldered to a four-pinned female USB A port, this port also includes an IP67 rated sealing kit by means of gaskets and a cap (which is not present in the pictures). Rubicon case had to be modified to allow the installation of both the button and the USB port, in both installations rubber bands have been used to maintain the water resistance.

In figure 3.9, the interior of a GNSS/Hybrid sensor can be observed. In figure 3.9(a), the placement of the antenna is shown. As it can be noted it is isolated from the rest of the components by a 3 millimeter aluminum plate which acts as a electromagnetic interference (EMI) shielding and as a ground plane increasing the performance of the antenna. In figure 3.9(b), the location of the rest of the components can be seen, it should be noted that the interior of the core module is painted with copper paint for also EMI-shielding purposes, the hole through which the antenna cable passes is also EMI-sealed with aluminum foil. For more information about the sensor case design regarding its electromagnetic compatibility, see 3.3.3.2. The Raspberry Pi Zero W board is fixed directly in the base sensor tape as well as the battery which thanks to its curved geometry, it is partially overlaid with the raspberry. This battery model shows its battery charge percentage by means of a tiny digital display which is activated by motion. Hence, the battery was placed in such a way that an operator can check the sensor charge by looking through the base tape. Finally every component connection has been made by means of ultra thin flat ribbon USB cables except for the connection between the USB charge port and the battery which has been carried out by a conventional USB cable.

(a) Side view.

(b) Front view.

Figure 3.8: GNSS/Hybrid sensor prototype exterior.



(a) Antenna placement. (b) Control board, battery and SDR placement.

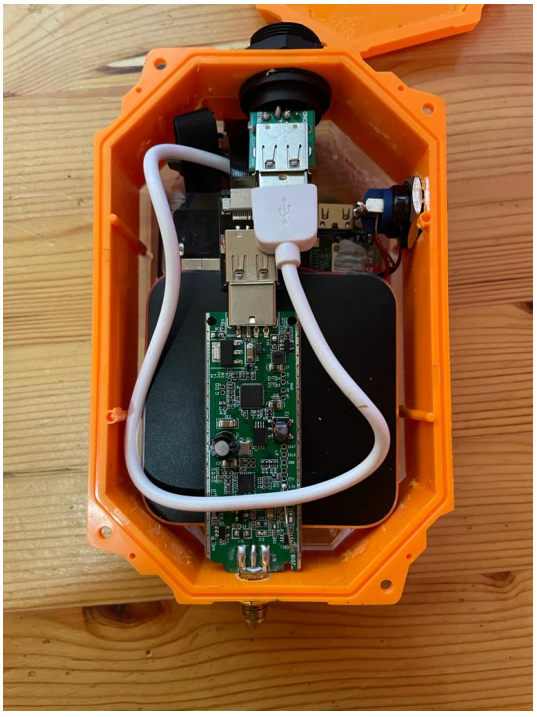
Figure 3.9: GNSS/Hybrid sensor prototype interior.

3.3.2 Cellular sensor prototypes.

A picture of a Cellular sensor prototype is shown in figure 3.10. As it can be deducted, Cellular sensors are very similar to GNSS sensors since they use the same hardware with the exception of the antenna. The main difference is that cellular sensors do not include the 3 mm. plates GNSS sensor had, this is because Cellular sensors don't suffer of self-interference at cellular bands and the antenna is not even placed in the interior of the case, instead of that, it is attached to the SMA port of the SDR which is accessible at the back of the sensor as it can be seen in figure 3.11(a). The interior of the sensor (figure 3.11(b)) follow the same distribution as in GNSS/Hybrid sensors except for the copper paint which in cellular sensors is not needed. Reset button and USB charge port are installed too as in GNSS/Hybrid sensors. The main advantage of this design is that, in terms of hardware, this sensors can be easily converted to GNSS sensors by changing the antenna.

Figure 3.10: Cellular sensor prototype exterior. Side view.

(a) Back view.



(b) Control board, battery and SDR placement.

Figure 3.11: Cellularsensor prototype back and interior.

3.3.3 Challenges and adopted approaches

In this section, most relevant encountered challenges will be defined, as well as the mechanisms performed in order to surpass them.

3.3.3.1 Rubicon case problems

In the early stages of the sensor design, a single unit of the sensor case was acquired from open.h company in order to check the product was adequate for the SNAP sensors approach. The quality and the unique features this product offered (such as a modular design, IP67 waterproof validation) made this product the selected one to be the exoskeleton of the prototypes. The only counterpart this box had was its reduced space, it was very difficult to find a high capacity battery with a compact design and with an adequate geometry to fit in this case. After several time, once the rest of components were acquired and their distribution within the case was set, a bigger order of several units of Rubicon case was made to Open.h who responded with a notification stating the product was discontinued and they had zero stock. Fortunately, after several searches, some units from a third party provider were obtained. But this Rubicon models were slightly different,they were labeled the as the exact same model and they were composed by the same modular elements as the tested unit but they were a 10% narrower in terms of height due to the fact that the two tape components were slightly different, a size comparison can be seen in figure 3.12. As a consequence, the stated design did not fit. Getting another model of case was not an option since several time was invested in figuring out how to fit every component inside this case and finding stock of the rubicon box with bigger tapes was impossible. Finally this problem was solved by filing the interior of the case and by using a super thin GNSS patch antenna, and ultra thin flat ribbon cables. With these modifications some millimeters of additional space were gained making possible the assembly of the GNSS prototypes.

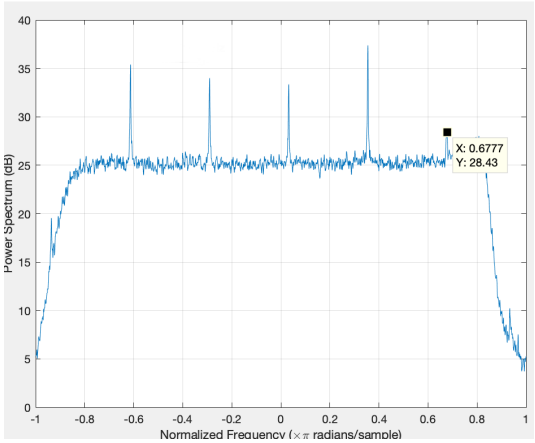
Figure 3.12: Sensor case ”Rubicon” tapes size comparison. left: latter units tape, right: first unit tape.

3.3.3.2 Electromagnetic compatibility of GNSS sensors

Once the first sensor prototype was assembled, the results of the performance tests were worse than the ones obtained with the proof of concept design. After further tests, it was concluded

that the performance degradation was related with the proximity of the sensor components, an electromagnetic interference near the L1 band generated by some component/s was affecting the sensor performance. In order to find the interference source, all components were distanced themselves using long wires, and then some tests were performed placing the antenna near to each of them, one by one. After that, the results were processed and visualized using a toolset called "testing tools" which are defined in section 4.1, furthermore, SDR# [20] software was also used in order to visualize the acquired spectrum in real time. During this process, some spurious were detected in the spectrum (figure 3.13) as well as performance degradation.

(a) Spurious present in the spectrum. Visualization using SDR#.



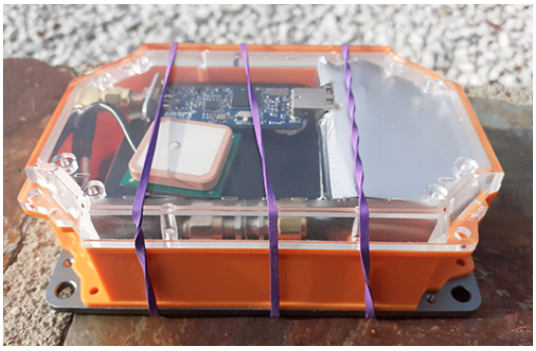
(b) Spurious present in the spectrum. Visualization using Matlab.

Figure 3.13: Electromagnetic interference visualization.

It was concluded that the present USB connections of the ribbon wires were the source of the existing interference. Replacing the ribbon wires was not an option due space restrictions. The purposed approach was to somehow isolate the antenna from the rest of the components. At this point it started a process of different electromagnetic shielding design iterations using diverse materials such as simple aluminum foil, a composite of plastic and several aluminum foil layers, 0.5mm aluminum plates and copper paint. Dozens of designs were tested by using SDR# and the "testing tools" toolkit. After this daily process, a test report was written for every test performed during the day including: timestamp, some pictures of the setup, IDs of the executions of the experiment, number of total executions launched, number of successful executions, a figure representing the acquired satellites and their carrier to noise ratio (C/N_0) level and a figure of the position errors obtained. Two examples of the electromagnetic interference shielding design iterations are shown in the figure 3.14.

thanks to the extensive testing phase, it was noticed that the 3 mm. aluminum plate of the Rubicon Box which were using as the sensor base, was acting as a reflector which was causing an interference enhancer. Taking this fact in account together with everything learned in this

process, it was decided to take in advance the modular design of the Rubicon box to change the case modules distribution, the result design is represented in figure 3.8. The central module of the case was painted with copper paint and the 3 mm. aluminum plate was modified to be as an electromagnetic interference shielding cover, some adaptations were made to it in order to place it in between of the core module and the upper tape. Finally a hole was drilled in order to pass the antenna wire through it. As a result, the antenna remains isolated from the rest of the components and moreover, the aluminum plate works as a mass plane which enhances the sensor performance. Hence, not only the problem was solved but also the sensor performance in terms of signal acquisition quality greatly improved.



(a) A cover made of two layers of paper and three layers of aluminum foil covering every connection of the sensor.

(b) A cover made of 0.5mm aluminum plate covering every connection of the sensor and part of the caseless SDR.

Figure 3.14: Two of the several EMI-shielding designs tried for the sensor prototype.

3.3.3.3 Raspberry Pi zero W writing speed bottleneck

Until the middle stages of the development phase of the project, RTL-SDR was the only RF front-end used. With the addition of the Airspy Mini model, problems were found when processing signal snapshots of more than 100 ms while when doing the same with an RTL-SDR everything was working fine. In order to figure out what was happening, some captures of a known signal were made. A signal generator was used to inject a 25 MHz sinusoid to the Airspy Mini directly through its SMA port. Then, a capture of 200 ms was made for the later visualization using Matlab. As it can be seen in figure 3.15, after 100 ms of capture random chunks of samples were being discarded. This was a not a problem for GPS signal captures since only 50 ms of signal is required by the platform, but for Galileo and Cellular signal captures around 200 ms is required. Finally, the problem cause was found:

- Raspberry Pi Zero W theoretical maximum memory writing is 24 MBps.
- Airspy mini performs signal captures at 6.0 Msps and every output sample is 4 bytes long.

Hence, at least 24 MBps of write speed is required in order to correctly save the output data of the Airspy Mini which is the theoretical maximum write speed of the Raspberry Pi Zero W. RTL-SDR only requires 6 MBps which is a fourth of what Airspy Mini requires, thats why long signal captures performed with the RTL-SDR were being correctly processed while the ones captured by the Airspy Mini were leading to errors.

After several tries, Airspy Mini Drivers were successfully modified in such a way that stable captures of more than 400 ms were achieved. Furthermore, different models of high performance SD cards were tested and every single sensor SD card was replaced with SanDisk Extreme SD cards. As a result, stable signal captures of more than 1 second were achieved.

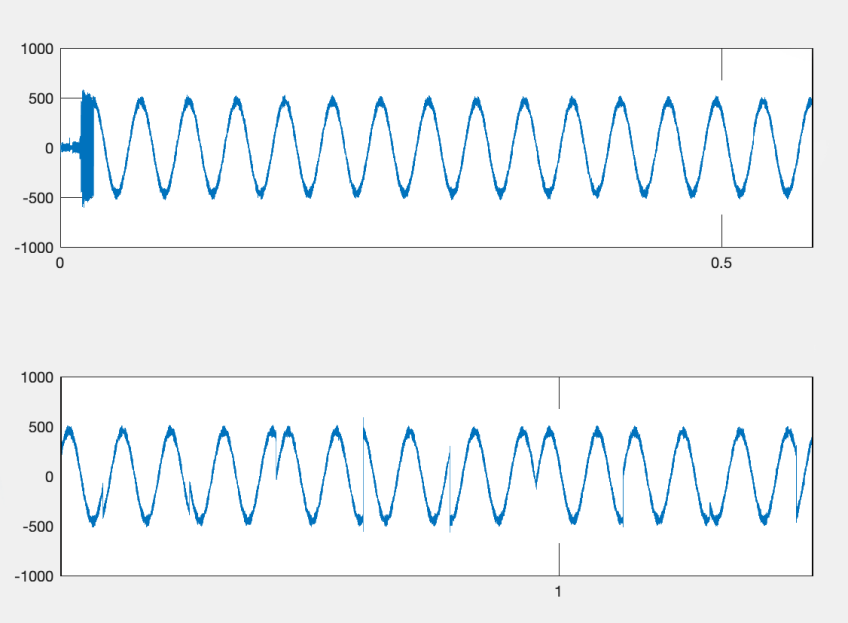


Figure 3.15: Visualization of 200 of a 25 MHz sinusoid captured usign an Airspy Mini attached to a Raspberry Pi Zero W usign original AirSpy drivers. From 0 to 100 ms (above). From 100 ms to 200 ms (below).

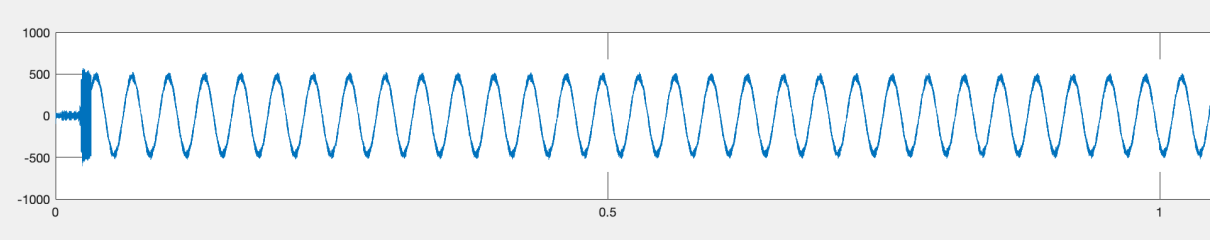


Figure 3.16: Visualization of 200 ms of a 25 MHz sinusoid captured usign an Airspy Mini attached to a Raspberry Pi Zero W after Airspy Mini drivers modification.

3.4 SNApp: Sensor software package.

Once the hardware was built and tested it was the time to develop a set of software routines that allowed the sensor to operate autonomously. SNApp is a software package installed in an operative system of a computing board (e.g., Raspberry Pi). It has two main tasks: First one is to control the radio-frequency (RF) front-end (e.g., software-defined radio) of the GNSS and/or cellular sensor in order to capture snapshots of real signals when requested. The software used for this goal is the front-end drivers required by the computer board and scripts for gathering GNSS and/or cellular signals and generating raw samples files when requested. Second one is allowing the sensor to communicate and interact with the Testbed (i.e., SNAP service), and it is achieved with the implementation of an API. The software package shall then satisfy the following set of which are listed in table 3.2.

Ref	Description
REQ-SNApp-01	The app SHALL periodically send requests to the Testbed to update its state.
REQ-SNApp-02	The app SHALL receive JSON files and update its state with the loaded parameters.
REQ-SNApp-05	The app SHALL configure the RF front-end of the sensor to work at the desired frequency band (i.e., GNSS and cellular).
REQ-SNApp-04	The app SHALL send JSON files with meta-data of the received signals and the configuration parameters of the execution.
REQ-SNApp-05	The app SHALL send binary files including raw samples of GNSS or cellular signals.

Table 3.2: SNApp requeriments.

3.4.1 SNApp components definition

Sensor files and folders structure can be found in the figure 3.17

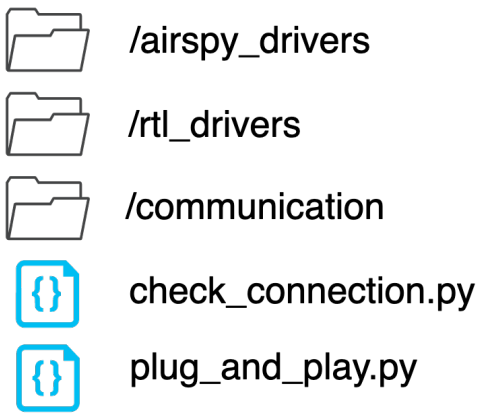


Figure 3.17: SNApp folders and files.

Next, a description of every file and folder is provided.

- `/airspy_drivers`
  Inside this folder Airsy Mini drivers and installation files can be found.

- `/rtl_drivers`
  Inside this folder RTL-SDRv3 drivers and installation files can be found.

- `/communication`
  This folder contains all necessary software the sensor needs in order to connect and operate with the SNAP service. For more information, see the following section.

- `plug_and_play.py`:
  This python file contains the sensor boot routine which is set by the use of `cron`. Once a sensor is registered in the platform, once it boots up it starts operating automatically. If the sensor is not already registered in the platform, it does nothing.

- `check_connection.py`:
  This python file contains a function that checks network and platform status. It is called in `plug_and_play.py`. Sensor will start working only if there is network connection and if the platform is online.

The `/communication` folder structure can be observed in the figure 3.18

Figure 3.18: `/communication` folders and files.

Next, a description of every file and folder is provided.

- `/data`
  This is the directory where temporary raw signal capture files are stored until they are sent to the SNAP service. After that they are deleted, hence this directory should remain empty.

- `msbCompressAirspy/msbCompressRTL`
  these C++ compiled files are the ones in charge of the Most-Significant-bit compression of the RTL-SDR and the AirSpy mini output samples files. They process samples files in such a way that only the most significant bit of every sample is taken and the rest of the bits are discarded. Their corresponding `.cpp` source code files can be found in RTL-sensor-workspace and airspy_host-master folders respectively.

- `configuration.json`
  This file includes all necessary parameters for the sensor to perform a signal capture. It is automatically created by the sensor registering process (`init.py`).

- `credentials.json`
  This file contains the user security token, the sensor id and the sensor type (GNSS/Cellular). The token is obtained and automatically written in this file in the registering process process (`init.py`).

  `get_frontend.py`
  This script is used to automatically detect which SDR the sensor is working with (RTL-SDRv3 or AirSpy Mini).

- `capture_signal.py`
  This script is used to gather and save signal. It contains the necessary functions that allow remote signal gathering. Once the script is called, the sensor will automatically identify the SDR model the sensor is using (by means of get_frontend.py), then it will capture a signal snapshot according to `credentials.json` parameters, sensor type and detected SDR model.

- `run.py`
  Executing this file starts the listening mode of the sensor. Sensors will execute this script automatically when booted up if internet connection is detected and HANSEL platform status is online by means of `plug_and_play.py`.

- `const.py, obtain_configuration.py`
  These files contain code information, variables or functions necessary for the other scripts to work. Its content should not be modified.

SNApp was built in a way that it handles everything needed to capture signal snapshots accordingly to the sensor type (GNSS/Celluar) and the attached SDR. It handles the Bias-Tee activation for both SDR models (it should be turned on for GNSS sensors and off for cellular sensor as the cellular antennas used are not active), the data compression, which is optional for GNSS and not allowed for cellular, and the SDR internal gain coefficient which since GNSS and cellular signal power is so different, GNSS sensors need the maximum gain and for cellular sensors SDR gain must be set to a much lower level. As a result, GNSS and cellular sensors share the same software package. This is very useful since a single device can be converted from GNSS to cellular sensor (or vice versa) with a single and fast software set and an antenna change.

3.4.2 SNApp workflow

SNApp workflow is composed by three stages: initialization, listening and calibration.
Initialization is required so that the sensor can perform signal captures and set up the communication with the external service. Once the sensor is initialized, by means of a listening routine, it

will check for configuration changes and pending position request (execution) every time period. Calibration is needed in order to get the characteristic error caused by the local oscillator when capturing at L1 band, depending on the method chosen it can be performed before or after of the listening stage. In the following sections these three sensor workflow stages are defined.

3.4.2.1 Initialization

The sensor initialization is based on the SNApp script `init.py` which has to be manually executed from the sensor (e.g. using SSH protocol). When executed, it asks the user to manually introduce the authentication token (available in the FBS platform "my Account section"), a name to identify the sensor and the sensor type (G/H/CP which stand for GNSS/Hybrid/Cellular Physic). As a result of the initialization, the sensor automatically registers itself in the external service and generates two files: `configuration.json`, which includes the default parameters to perform a signal capture, and `credentials.json`, that contains the required information in order to communicate with the external service.

The command that user has to manually execute from /home/pi/SNApp folder is shown below:

```
$ python3 init.py
```

Initialization workflow is depicted in the figure 3.19 and an example of a G-type sensor initialization process using `init.py` is shown in the figure 3.20.

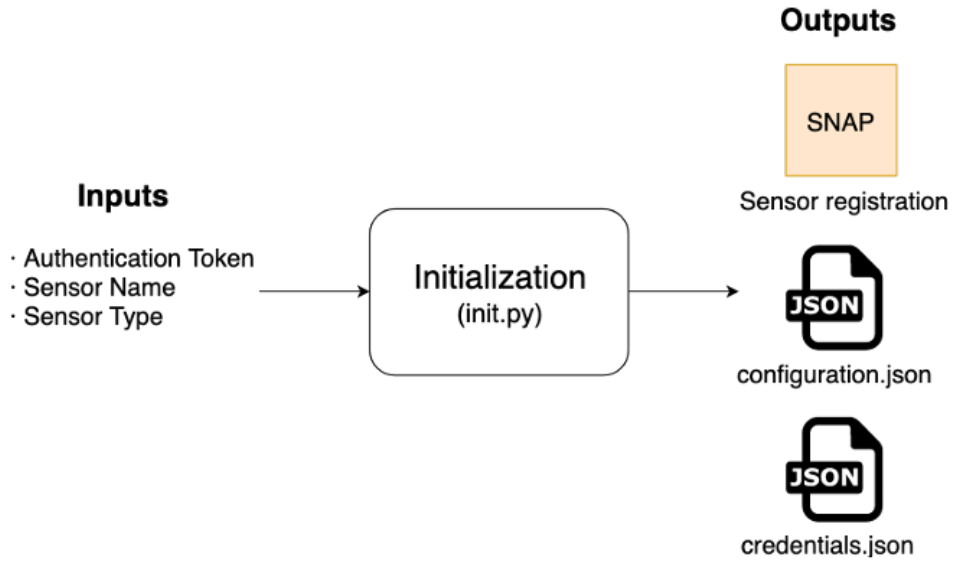


Figure 3.19: Sensor initialization block diagram

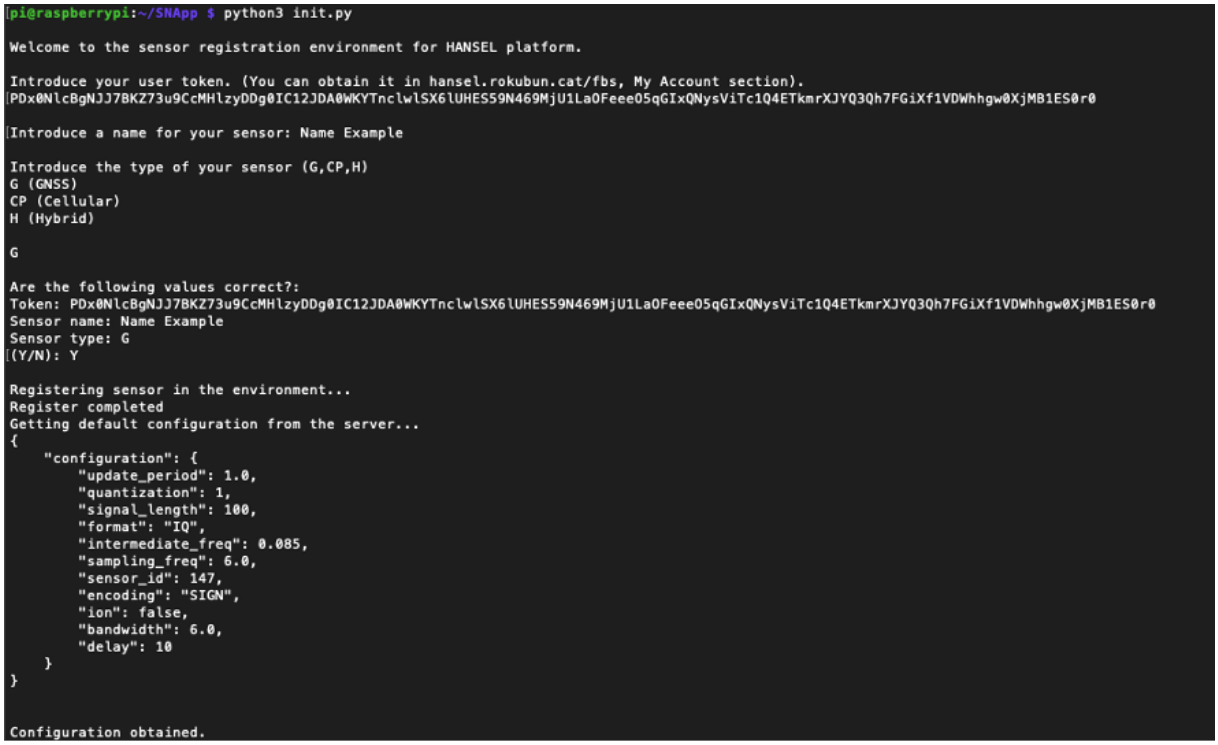


Figure 3.20: G sensor initialization logs

3.4.2.2 Listening

Once sensor has been correctly initialized, when it is booted up it will automatically run the listening routine by means of a `cron` task. Alternatively it can be manually executed by the user by manually running the command below from `home/pi/connection directory`.

```
$ python3 run.py
```

Sensor listening workflow is depicted in the figure 3.21. The actions the sensor performs during this routine are listed below:

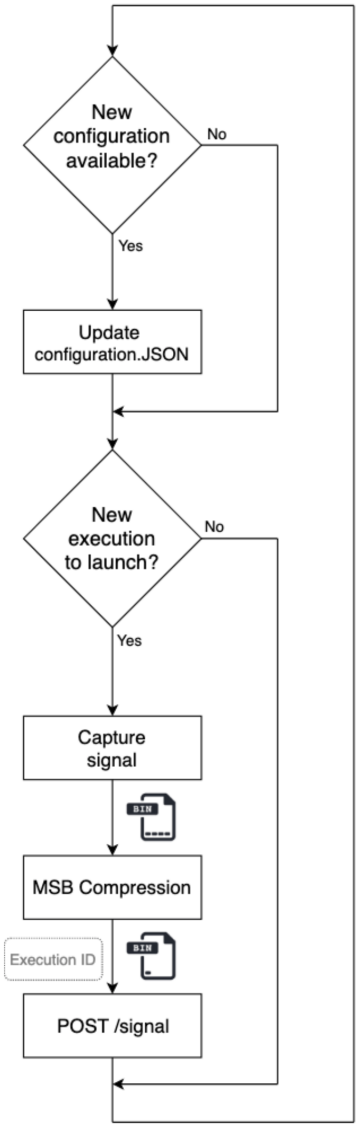


Figure 3.21: Sensor listening routine

1. Check whether there is a new configuration available. If a new configuration has been set, the sensor updates the local configuration file `configuration.json` with the new parameters. If there is not new configurations available, it does nothing.

2. Checks whether there is a pending execution to be launched. If there is, it performs steps 3 to 6. If there is not, directly jump to step 6.

3. Gather signals using `capture_signal.py` and according to `configuration.json` parameters and save the gathered signal in a local temporary binary file in `/data` directory.

4. Perform a compression of the temporary binary file using Most Significant Bit compression. This action is optional, it depends on the sensor configuration.

5. Obtain the execution identifier from the platform and send the binary file to SNAP's external service (CloudGNSSRx) to be processed.

6. Wait a predefined period of time and go back to step 1.

Once the sensor is listening, the registered user is able to send the desired configuration regarding the signal capture as well as position requests (executions). In figure 3.22 the logs of a sensor getting and applying a configuration request are shown, similarly in figure 3.23 it can be seen the logs of an RTL-SDR sensor getting and successfully performing an execution request with MSB data compression enabled.

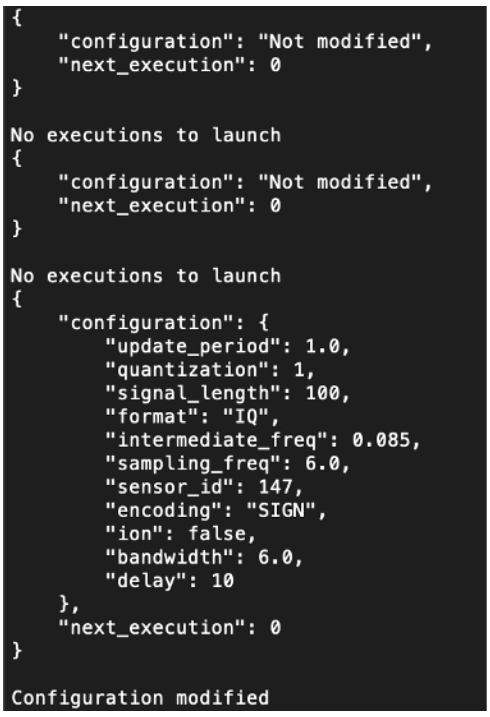


Figure 3.22: Listening routine sensor logs. (3 cycles, no executions, 1 successful reconfiguration)

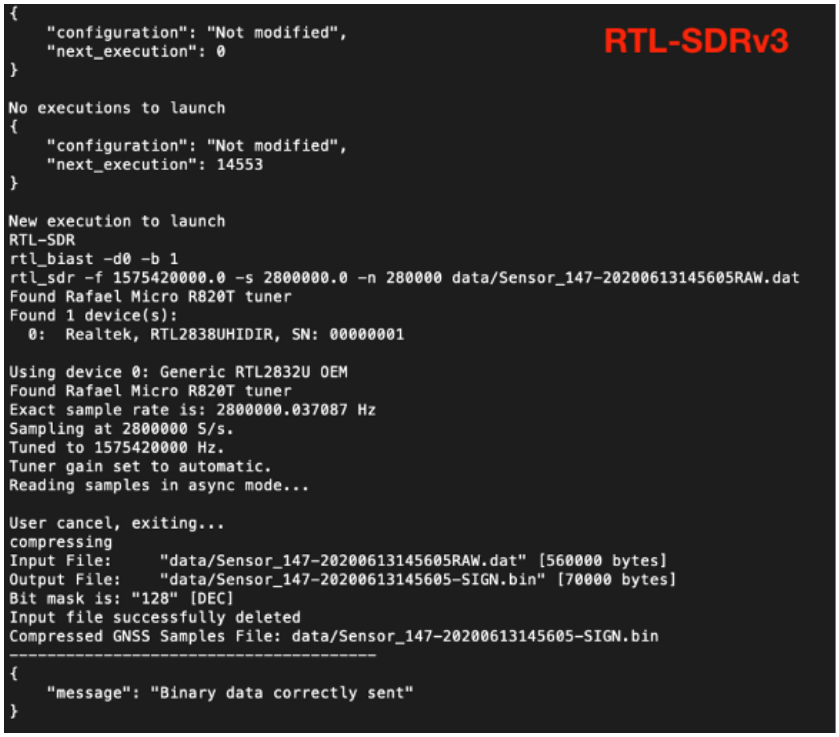


Figure 3.23: Listening routine sensior logs. (3 cycles, 1 successful execution performed, no reconfiguration)

3.4.2.3 SDR Calibration

Both the RTL-SDR and the Airspy Mini have a local oscillator based on a Temperature Controlled Crystal Oscillator (TCXO) with considerable frequency stability despite being low-cost devices (especially the RTL-SDR). However, for the operating frequencies of GNSS signals, around 1.5 GHz., only a few of error causes frequency errors of several kHz. This makes it difficult to acquire the visible satellites as the expected frequency to which the SDR has been tuned does not coincide with the real frequency to which the signal is centered. Therefore, a calibration mechanism is necessary to correct this local oscillator offset.

kalibrate-RTL tool

In the early stages of the project, only RTL-SDR front-ends were used. Every unit of RTL-SDR had its own characteristic error, so a calibration process had to be performed for each device. Since the error was constant under the the development scenario (indoors installations), it only was necessary to perform this process once per device. In order to solve this problem, kalibrate-RTl [18] tool was used.

Kalibrate-RTL, can scan for GSM base stations in a given frequency band and can use those GSM base stations to calculate the local oscillator frequency offset. This tool was very useful for the proof-of-concept stages of the project, when accuracy was not the main objective but it had two disadvantages:

1. Need for the presence of nearby GSM base stations.
2. Only supported by RTL-SDR devices.
3. The process takes few minutes.

Calibration using GPS dopplers

As far as clock error is concerned, Airspy Mini units experience a similar phenomenon and despite the fact the manufacturer assures they present a maximum of ±0.5ppm, during the development of this project some errors greater than 1.5 ppm were detected. Furthermore, the error clock coefficient registered during the outdoors experimentation with both RTL-SDR and Airspy Mini models, turned to be highly temperature dependent which could vary depending not only on the weather, but also by the frequency of performed executions (due to the heat generated by themselves). Hence, the calibration process had to be performed periodically in order to assure good position estimations. Taking into account all the mentioned above, it was decided to develop a tool to calibrate any model of SDR.

GPS satellites complete an orbit around the Earth every 12 hours or so, which means they travel at a constant speed of about 14.000 km/h. This velocity generates a Doppler effect that the user sees through its radial component on the direct line of sight between the satellite and the user. Since the paths that the satellites follow are all different, seen from the user receiver, the Dopplers with which the receiver receives each of its signals are also different. These Dopplers are known since they depend on the orbit of the satellites and the position of the user, although it is not necessary for the latter to be very precise for the intended purpose.

To calculate the Dopplers, the open-source orbital simulator SGP4 [16] has been used which, from a file of TLE orbital ephemerides (*Two Line Element*) available for example at [15], and an approximate position of the user, provides the expected Dopplers at that position for a given date and time. Once the expected Dopplers are known, the remote server receiver can be configured to search the satellites for different tentative Doppler values. Once the search is finished, the sensor clock offset ends up being the difference between the Doppler found by the remote receiver and the expected Doppler. This calibration can be carried out periodically in case the sensor is subject to temperature variations that cause the offset of the local oscillator to vary. When calibration is not needed, the remote receiver can disable the Doppler search and directly use the values provided by the SGP4 simulator plus the calibrated offset.

3.5 User-Sensor interaction

In this section, operator-sensor (or user-sensor) interaction is going to be explained from the very basis like how to access the sensor files and command prompt or to how the platform validation works, to how sensor position requests and results are carried out. Note that despite the fact that there are 3 types of sensors (GNSS, Cellular or Hybrid), this section will take GNSS sensors as an examples since from a operational point of view user-sensor interaction is the same with some exceptions which are mentioned.

3.5.1 Manually interacting with the sensor

In order to manually interact with the sensor e. g. in order to manually execute commands, since control boards run a LINUX-based operating system, Secure Shell protocol (SSH) can be used. Assuming the sensor is connected to the user network and that the sensor IP is known, depending on the operating system the user equipment runs, there are different recommended options.

- **LINUX/MacOS users**
  A connection with the sensor can be established by executing the following command in the command shell.
  `python3 user@xxx.xxx.xxx.xxx`
  where `xxx.xxx.xxx.xxx` is the IPv4 address of the sensor board which depends on the network the sensor is connected to. After the execution of this command, a password will be asked to be manually introduced by the user.

- **Windows users**
  If user equipment runs Windows, "PuTTY SSH client" [19] can be used, available in https://www.putty.org/. In order to establish a connection with the sensor using PuTTY, user must introduce the information as indicated the figure 3.24.
  where `xxx.xxx.xxx.xxx` is the IPv4 address of the sensor board which depends on the network the sensor is connected to. After the execution of this command, a password will be asked to be manually introduced by the user.

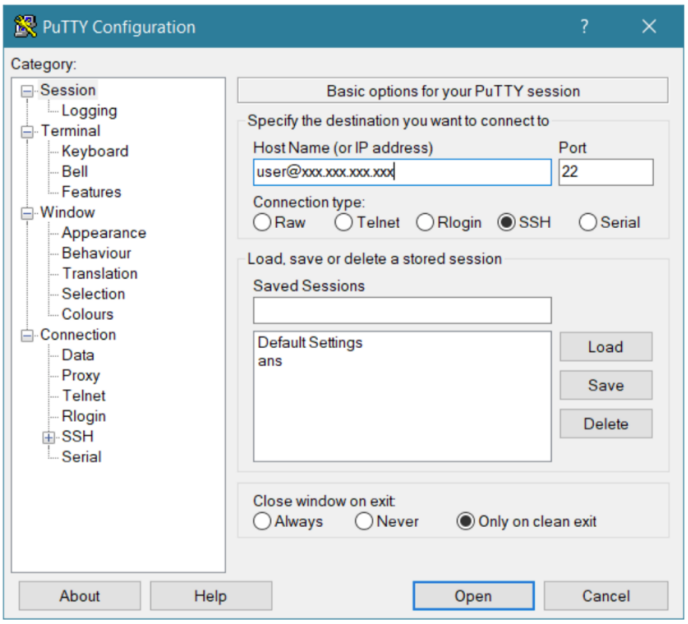


Figure 3.24: Putty tool.

Once the user has successfully accessed the sensor via ssh or PuttY, he can manually initialize the sensor in the platform by means of the initialization script `init.py` and can also start the listening routine by means of `run.py`. Once the sensor is already registered, a booting routine can be enabled to automate this process by means of accessing the sensor crontab list (figure 3.25) and uncommenting the last line. This way, sensors start working automatically when booted up.

```
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
@reboot sudo python3 /home/pi/plug_and_play.py # > /home/pi/Desktop/log.txt
```

Figure 3.25: Crontab list with the booting routine enabled.

3.5.2 User scripts

As it is explained in previous sections, the user is able to remotely request a sensor for reconfigurations and signal captures for the later processing. The tool which is in charge of that is called FBS, which uses the SNAP API as communication interface. FBS development was in charge of an external company and first operational version came up at the very end stages of the project. In order to test the correct functionality of the sensors communication with the platform and the SNAP API itself, a set of python 3.7 scripts were developed to simulate the functionalities of the FBS service. In order to use them, user must have python3 with "requests" module previously installed.

User scripts files structure is depicted in figure 3.26. It is divided in four sections which are listed and briefly defined below:

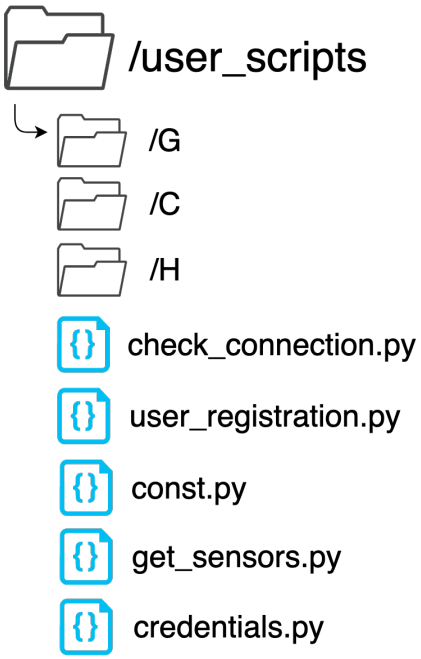


Figure 3.26: User scripts files structure.

- **/G:** This folder contains the set of scripts which are specifically used for SNAP G service.
- **/C:** This folder contains the set of scripts which are specifically used for SNAP C service.
- **/H:** This folder contains the set of scripts which are specifically used for SNAP H service.
- **Common scripts:** This set of scripts are shared by scripts contained in /G, /N, and /H folders.

A brief description of every common script is listed below:

- `check_connection.py:`
  This python file contains a function that checks network and platform status.

- `credentials.py`
  This file is where the user security token is stored. It is needed to communicate with the platform.

- `const.py`
  This file contains different parameters which are called every user script. User should never modify any parameter of this file.

- `user_registration.py`
  This script sends asks the user to manually type the required data in order to be registered in the platform and sends it to the cloud, cloud will respond with the user security token which user must put in `credentials.py`. Once this process is completed, user should wait for the validation email.

- `get_sensors.py`
  This script returns all sensors registered by the user. It is used to facilitate the sensor selection process which is present in several functionalities.

3.5.2.1 SNAP G scripts

SNAP G folder structure is depicted in figure 3.27

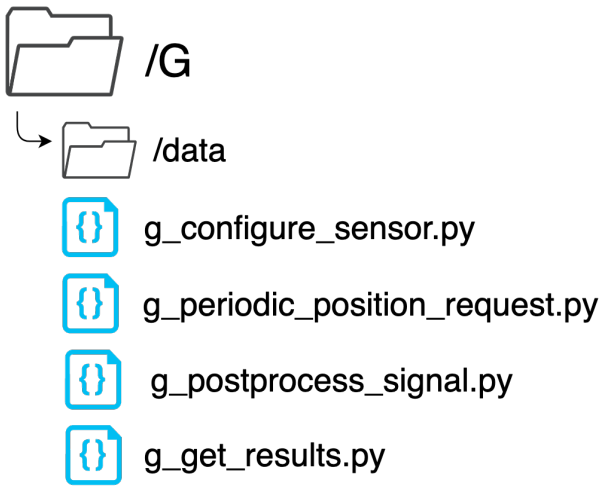


Figure 3.27: /G folder structure.

A brief description of every file and folder is listed below:

- `/data`
  This folder is where signal files to be post processed are stored.
- `g_configure_sensor.py`
  User should use this script in order to change any configuration parameter of GNSS sensors, user never should manually edit the sensor configuration.json internal file, doing it would cause mismatches between the configuration of the sensor and the cloud that would lead to errors. To correctly use this script, user should open it and edit the desired parameters, save changes and execute it. For more information, see section 3.5.4.
- `g_periodic_position_request.py`
  This script requests the API to configure the sensor in such a way that it periodically captures a snapshot of RF samples and sends it to the platform. For more information, see section 3.5.5.
- `g_postprocess_signal.py`
  This script sends an already existing file with RF samples, available at the user's computer, to the API for its processing at the cloudGNSSrx platform. For more information, see section 3.5.6.
- `g_get_results.py`
  This script queries the API to retrieve the results of a desired execution which has been performed by a GNSS sensor. For more information, see section 3.5.7.

3.5.2.2 SNAP C scripts

SNAP C folder structure is depicted in figure 3.28

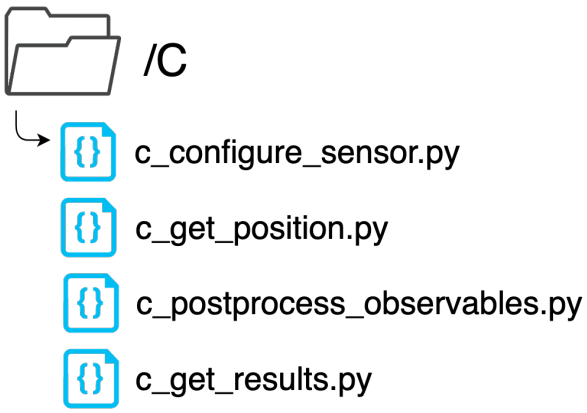


Figure 3.28: /C folder structure.

A brief description of every file and folder is listed below:

- `c_configure_sensor.py`
  User should use this script in order to change any configuration parameter of cellular sensors, user never should manually edit the sensor configuration.json internal file, doing it would cause mismatches between the configuration of the sensor and the cloud that would lead to errors. To correctly use this script, user should open it and edit the desired parameters, save changes and execute it.

- `c_get_position.py`
  This script is used to request a position estimation using SNAP C service, user must attach a KML file containing the near cellular base stations and their positions.

- `c_postprocess_observables.py`
  This script sends an already existing file with RF samples, available at the user's computer, to the SNAP C service in order to be processed and get some data parameters about the detected base stations. No position is returned.

- `c_get_results.py`
  This script queries the API to retrieve the results of a desired execution which has been performed by a Cellular sensor.

3.5.2.3 SNAP H scripts

SNAP C folder structure is depicted in figure 3.29

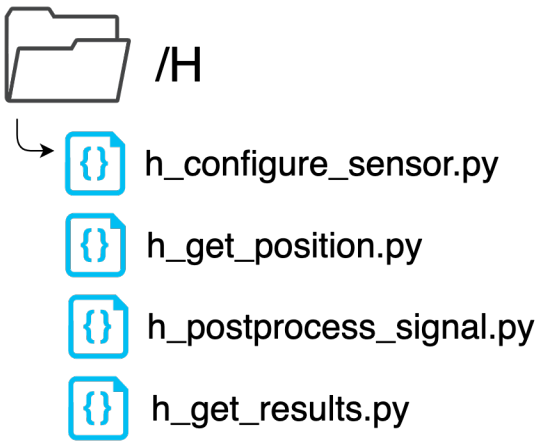


Figure 3.29: /H folder structure.

A brief description of every file and folder is listed below:

- `h_configure_sensor.py`
  User should use this script in order to change any configuration parameter of cellular

sensors, user never should manually edit the sensor configuration.json internal file, doing it would cause mismatches between the configuration of the sensor and the cloud that would lead to errors. To correctly use this script, user should open it and edit the desired parameters, save changes and execute it.

- `h_get_position.py`
  This script is used to request a position estimation using SNAP H service, user must attach a KML file containing the near cellular base stations and their positions.

- `h_postprocess_signal.py`
  This script sends an already existing file with RF samples, available at the user's computer, to the SNAP H service in order to be processed and get a position estimation.

- `h_get_results.py`
  This script queries the API to retrieve the results of a desired execution.

3.5.3 User registration

In order to register and use a sensor with the SNAP platform, the user has to be previously registered and validated by an admin. In order to do so, user must execute the following command:

```
$ python3 user_registration.py
```

After that, user must follow the instructions in the screen. An example of a successful user registration is depicted in figure 3.30

```
$ python3 user_registration.py

Welcome to the user registration environment.

Introduce your name:
>Quim Ganez

Introduce your email:
>Joaquim.ganez@e-campus.uab.cat

Introduce a password:
>**********

Introduce the name of the organization:
>UAB

Introduce the purpose for this account:
>Testing

Registering the user in the platform...
User correctly registered.

This is your current token:
PDx0NlcBgNJJ7BKZ73u9CcMHlzyDDg0IC12JDA0WKYTnclwlSX6lUHES59N469MjU
1LaOFeeeO5qGIxQNysViTc1Q4ETkmrXJYQ3Qh7FGiXf1VDWhhgw0XjMB1ES0r0

Save it and put it in the TOKEN field from credentials.py
whenever you want to use this user again. Do you want to exit?
(Y/N)
>Y

A validation e-mail will be sent to: Joaquim.ganez@e-
campus.uab.cat
Exiting the environment...
```

Figure 3.30: Successful user registration using `user_registration.py`. (User terminal logs)

Once the process is completed, user must wait for the validation email before trying to register a new sensor in the platform.

3.5.4 Configuring the sensor

For the user to be able to configure a sensor, it must be previously registered in the platform and running the listening routine. The user must open the `configure_sensor.py` file which contains a json list with all the configurable parameters. User must manually edit the desired parameters, save the file and execute the following command:

```
$ python3 g_configure_sensor.py
```

In figure 3.31, an example of a JSON configuration list is shown.

```
{
  "sensor_id": 320,
  "sampling_freq": 2.8,
  "format": "IQ",
  "update_period": 1.0,
  "intermediate_freq": 0.723,
  "delay": 10,
  "bandwidth": 2.8,
  "encoding": "SIGN",
  "quantization": 1,
  "signal_length": 100
}
```

Figure 3.31: JSON sensor-configuration list in `configure_sensor.py`

A brief description of every field is listed below:

- **sensor_id:** identification number of the sensor which is desired to be configured.
- **sampling_freq:** sampling frequency for the signal capture. [Mhz]
- **format:** format of the SDR output samples. ["SIGN" if MSB compression is desired, "INT" if not]
- **update_period:** time sensor must wait before checking for next configurations and executions. [s]
- **intermediate_freq":** frequency shift error caused by the SDR internal clock error at 1575.42 MHz. [MHz]
- **delay:** time fraction of signal to be discarded by the receiver. [ms]
- **bandwidth:** bandwidth of the signal capture. [Mhz]
- **encoding:** samples encoding. INT for not compressed samples, SIGN for compressed samples.
- **quantization:** size of samples. [b.]
- **signal_length:** desired total length (time) for the signal capture. [ms]

3.5.5 Requesting executions

As in sensor configuration requesting, for the user to be able to request a sensor for executions, it must be previously registered in the platform and running the listening routine. The user must open the `periodic_position_request.py` and modify the execution configuration JSON

list as desired. It consists in a set of parameters which are used by the CloudGNSSRx receiver to process the signal captures. An example of an execution configuration is depicted in 3.32.

```
{
"band": "1",
"system": "GPS",
"coh_time": 20,
"num_noncoh": 1,
"interference": 0,
"manual_sat_search": 0,
"assisted_dopp": 1,
"generate_agnss": 1,
"sat_list": [1,3,7,13,21,21,29,31],
"ref_pos": [41.4996264, 2.11289406, 139]
}
```

Figure 3.32: JSON execution configuration list in `g_periodic_position_request.py`.

A brief description of every parameter is listed below:

- **band:** as the CloudGNSSRx is a multi-band receiver, the band in which the snapshot has been captured must be specified. ”1” for GPS L1 C/A and Galileo E1C, ”5” for GPS L5/Galileo E5a.
- **system:** as the CloudGNSSRx is a multi-constellation receiver, GNSS system must be specified. User can choose between GPS and Galileo.
- **coh_time:** coherent correlation time. [Mhz].
- **num_noncoh:** number of non-coherent integrations.
- **interference:** interference detection flag. The CloudGNSSRx has an interference detection service which is triggered for an execution if this flag is set to ”1”.
- **manual_sat_search:** manual satellite search flag. If set to 1, CloudGNSSRx will search for every satellite in `sat_list` in a non-assisted way.
- **assisted_dopp:** assisted satellite search flag. If set to 1, manual_sat_search must be set to 0 and vice versa.
- **generate_agnss:** automatic assistance data generation flag. if set to 1, CloudGNSSRx will use an orbital simulator generate assistance data.
- **sat_list:** list of visible satellites. Only needed if manual_sat_search flag is set to 1.
- **ref_pos:** Reference position latitude, longitude and height coordinates. Has to be ± 75 km. accurate to solve the navigation equations. In testing operations, should be the true position so position error becomes the actual error between the estimated position and the true one.

Once the user has edited the desired execution parameters, he must save the file and execute the following command:

```
$ python3 periodic_position_request.py (<timePeriod>) (<numberOfRequests>)
```

where the parameter `timePeriod` defines the time between signal capture requests, and `numberOfRequests` defines the total number of position requests (executions) to be launched. They are optional arguments. The command `periodic_position_request.py` accepts 0, 1 and 2 arguments. Depending on the number arguments it performs different actions:

- No arguments: a position will be requested to the sensor every 60 seconds until user manually interrupts the script.
- 1 argument: a position will be requested to the sensor every `timePeriod` seconds until user manually interrupts the script.
- 2 arguments: a position will be requested to the sensor every `timePeriod` seconds until the number of requested executions reaches `numberOfRequests`.

An example of the successful request of a single execution is shown in the figure 3.33.

```
$ python3 periodic_position_request.py 1 1

Showing available sensors:
125  221  320

Introduce the id of the sensor you want to get the position from:
>320

Two arguments provided. One request will be made every 1 second/s
until reaching 1 request(s).

{
  "execution_id": 10320
}
```

Figure 3.33: Successful position requested to sensor 320 using `periodic_position_request.py`. (User terminal logs).

3.5.6 Post processing executions

SNAP platform allows the user to process a snapshot file without the needs of a sensor, it can be done using the `postprocess_signal.py` script. In order to do so, user must specify the signal capture parameters (sensor configuration) and the execution configuration, both configurations are represented as JSON lists inside the script and are exactly the same format as can be seen

in figure 3.31 for sensor configuration and 3.32 for execution configuration.
Once the user has edited the desired parameters, he must save the file and run the following command:

```
python3 post_process.py (<fileName>)
```

where fileName is the name of the snapshot file to be processed. It must be located in the `/data` file. Once executed, the platform will respond with the ID of the execution.

3.5.7 Retrieving results

As it is mentioned in previous sections, SNAP platform has the ability to store multiple data regarding the executions, included the position estimation. If the user wants to check the estimated position of an execution, he can do it by using the `extract_results.py` by executing the following command:

```
python3 get_results.py (<executionID>)
```

Where executionID is the identification name of the execution whose result is going to be retrieved. Once executed, the platform will respond with the estimated position in latitude, longitude, altitude coordinates. An example of a successful retrieval of the position estimation of an execution is shown in the figure 3.34.

```
$ python3 get_results 10320

The position estimation of the execution 10320 is:
[41.4996264, 2.11289406, 139]
```

Figure 3.34: Successful retrieval of the position estimation result of execution 10320 using `get_results.py`. (User terminal logs)

Chapter 4

SDR sensor prototype validation

In this chapter the process of testing needed to validate the sensors is explained, as well as the tools which were used to facilitate the process.

4.1 Testing tools

During the sensor developing phase of the project, testing the sensor was very frequent. In order to check the correct operation and the performance of the sensor, sending executions and visualizing the results was a daily task. The visualization and comparison of results data was a very tedious task even for a couple executions and it was unthinkable for large datasets of hundreds of executions. With the motivation of avoiding this problem, the development of a software package called "testing tools" started. It consists in a set of python 3.7 and Matlab scripts which can be distinguished in two parts. First one bring the ability to retrieve from the platform large datasets of SNAP G execution results. The second one automates the needed processing for the visualization of some parameters of interest regarding the retrieved datasets.

As it can be observed in figure 4.1, testing tools set is divided in three parts:

- `/data_retrieval`
  This folder contains a set of python 3.7 scripts with which an admin user is able to retrieve datasets of G/H execution results from SNAP DB. Once retrieved, they are stored in `/data` folder.
- `/data_processing`
  This folder contains a set of matlab scripts with which an user can process datasets for the visualization of some values like acquired satellies and carrier to noise (C/N_0) values.
- `/data`

This folder is the directory where the output files of data retrievals are stored.

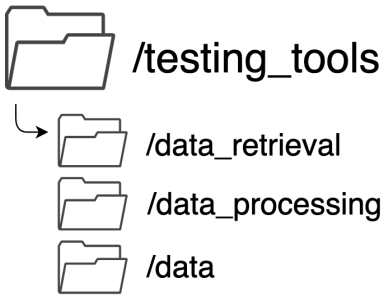


Figure 4.1: Testing tools files structure.

In the following sections a more in-depth description of `/data_retrieval` and `/data_processing` is made.

4.1.1 Data retrieval

This tool set main functionality is to allow an admin-level user to extract results data in a bulk manner. Its file structure is depicted in figure 4.2.

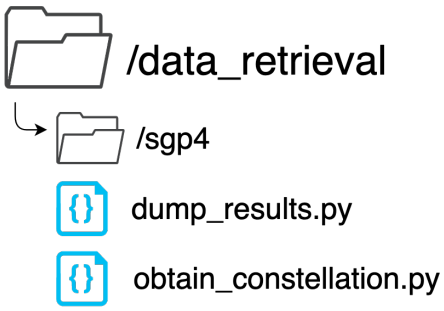


Figure 4.2: `/data_retrieval` files structure.

A brief description of every file and folder is listed below:

- `/SGP4`
  This folder contains all the dependencies that `obtain_constellation.py` needs to operate.
- `dump_results.py`
  This file is in charge of retrieving the necessary data from SNAP DB.
- `obtain_constellation.py`
  There are some parameters of interest which are not stored in SNAP DB such as GNSS assistance data, this script is based in the same tool which is used in the CloudGNSSrx

to obtain the GNSS assistance. Given the user position and time, this tool calls an online orbital simulator and returns the list of visible satellites, their doppler respect to the user, elevation and azimuth.

Several modifications had to be made to `obtain_constellation.py` for its integration in this toolset: firstly, since it was coded in python 2 it had to be translated to python 3.7. Secondly, original tool output was a .txt file whose format is depicted in figure 4.4 which is convenient for visualization purposes but it does not fit well for the desired functionality of this toolset. Hence, it had to be heavily modified to give it a modular shape that would fit into the toolset. The output of the modified script is depicted in figure 4.3, this way instead of returning a text file, it returns an array of JSON objects which include all the information in an orderly and easy to deal with way. The assistance data generated with `obtain_constellation.py` is added to the execution data which is retrieved from SNAP DB with. As a result, for every execution, a JSON including SNAP DB and assistance data is returned. The returned data regarding of individual execution of an extracted dataset of 100 executions using `deump_results.py` is depicted in figure 4.5.


```
"assistance_parameters":[
{
"prn": "PRN 16",
"doppler": 1552.504223515738,
"elevation": 60.21960266704813,
"azimuth": 311.79517438804106
},
{
"prn": "PRN 21",
"doppler": 332.0038233908808,
"elevation": 77.97897104073621,
"azimuth": 209.3656962802267
},
{
"prn": "PRN 31",
"doppler": -3417.6669687841054,
"elevation": 15.787194656454375,
"azimuth": 195.45630608477745
},

.
.
.

{
"prn": "PRN 18",
"doppler": -2019.937939077457,
"elevation": 52.58890779318967,
"azimuth": 52.36580636407734
}]
```


Figure 4.3: `obtain_constellation.py` output JSON once modified.

```
GPS Satellites
------------------------------------------------
PRN 16 1552.50  60.21960266704813  311.79517438804106
PRN 21  332.00  77.97897104073621  209.3656962802267
PRN 31 -3417.66  15.787194656454375  195.45630608477745
.
.
.
PRN 18  -2019.93  52.58890779318967  52.36580636407734
```

Figure 4.4: `obtain_constellation.py` original .txt output file. Data order: Satellite PRN, user relative doppler, elevation and


```
{
"id": 16661,
"prn": [16, 21, 31, ... 18],
"cn0": [44.1957583, 43.5005748, 45.7115792, ... , 44.158077],
"sensor_pos": [41.2026563, 1.55297735, 69773.7234],
"dopp": [1408.99675, 213.971058, -3542.53189, ... , -2138.76101],
"assistance_parameters":[
{
"prn": "PRN 16",
"doppler": 1552.504223515738,
"elevation": 60.21960266704813,
"azimuth": 311.79517438804106
},
{
"prn": "PRN 21",
"doppler": 332.0038233908808,
"elevation": 77.97897104073621,
"azimuth": 209.3656962802267
},
{
"prn": "PRN 31",
"doppler": -3417.6669687841054,
"elevation": 15.787194656454375,
"azimuth": 195.45630608477745
},

.
.
.

{
"prn": "PRN 18",
"doppler": -2019.937939077457,
"elevation": 52.58890779318967,
"azimuth": 52.36580636407734
}]
}
.
.
.
"total_executions": 100,
"failed": 0,
}
```


Figure 4.5: `dump_esults.py` script JSON output for a single execution (id: 16661). Execution data (blue), assistance data (orange)

4.1.2 Data processing

This toolset provides processing and visualization functionalities for datasets created using `dump_results.py`. Its file structure is depicted in figure 4.6.

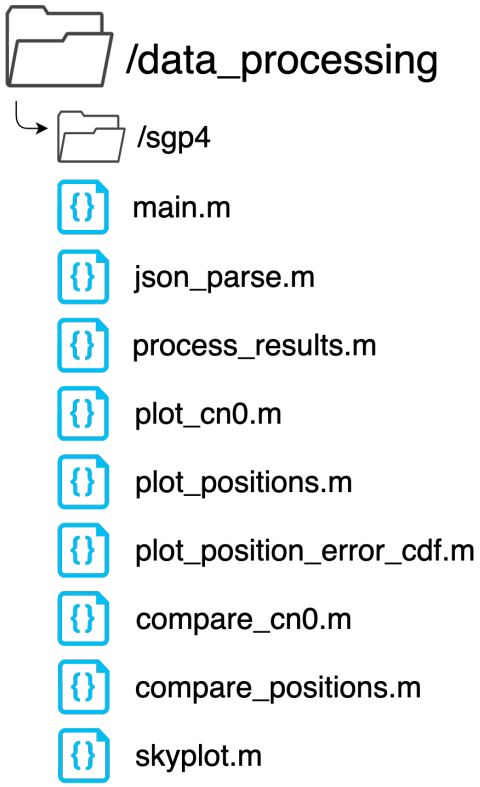


Figure 4.6: `/data_processing` files structure.

A description of every script is present in the following sections.

4.1.2.1 main.m

This is where the rest of the scripts must be called.

4.1.2.2 json_parse.m

- **Input arguments:**
  `file_path` (String)
- **Output:**
  `jsn` (Struct)
- **Description:**

Uses the indicated `file_path` to get the data json file retrieved from the platform and to load it into the matlab structure `jsn`.

4.1.2.3 process_results.m

- **Input arguments:**
  `jsn` (Struct).
- **Output:**
  `results` (Struct)
- **Description:**
  It gets the indicated jsn structure (which must be the output of `json_parse.m script`) and performs some processes in order to generate the results structure, for more information about the `results` output structure.

4.1.2.4 plot_cn0.m

- **Input arguments:**
  `results` (Struct), `bar_color` (Array), `figure_title` (String).
- **Output:**
  Figure.
- **Description:**
  It uses the results struct data obtained with the `process_results.m` script to represent the C/N_0 mean value of each satellite acquired as a bar graph. Furthermore, the maximum and minimum values of C/N_0 for each acquired satellite are also represented as a error bar. The bar graph color is defined by the input argument `bar_color` vector and the figure title is set by the input argument `figure_title` string.

4.1.2.5 plot_positions.m

- **Input arguments:**
  `results` (Struct),`dot_color` (Array),`figure_title` (String).
- **Output:**
  Figure.
- **Description:**
  It uses the results struct obtained with the `process_results.m` script to graphically represent the horizontal position errors with respect to a reference position (which has had

to be previously set by the user in `process_results.m`) using a scatter graph. The dots color is defined by the input parameter `dot_color` vector and the figure title is set by the input parameter `figure_title` string.

4.1.2.6 plot_position_error_cdf.m

- **Input arguments:**
  `results` (Struct), `trace_color` (Array), `figure_title` (String).
- **Output:**
  Figure.
- **Description:**
  It uses the `results` struct data obtained with the `process_results.m` script to represent the cumulative distribution function of the horizontal position errors. The trace color is defined by the input parameter `trace_color` vector and the figure title is set by the input argument `figure_title` string.

4.1.2.7 compare_cn0.m

- **Input arguments:**
  `results_vector` (Struct Cell), `bar_colors` (Array Cell), `legends` (String Cell), `figure_title` (String).
- **Output:**
  Figure.
- **Description:**
  This script is intended for the comparison of the C/N0 values of two or more experiments. The `results_vector` array (cell) must be filled with the desired `results` structures to be compared, these results structures have had to be previously obtained with the `process_results.m` script. The comparison consists in a grouped bars graph representing the C/N_0 values of each experiment. The values of each experiment or `results`, are distinguished by colors which are set in the `bar_colors` input argument and by their corresponding text legends which are specified in the legends input argument. The name of the resulting graph is set by the `figure_title` input argument.

4.1.2.8 compare_positions.m

- **Input arguments:**
  `results_vector` (Struct Cell), `dot_colors` (Array Cell), `legends` (String Cell),

`figure_title` (String).

- **Output:**
  Figure.
- **Description:**
  This script is intended for the comparison of the position error values of two or more experiments. The `results_vector` array (cell) must be filled with the desired `results` structures to be compared, these results structures have had to be previously obtained with the `process_results.m` script. The comparison consists in scatter graph. The values of each experiment or `results`, are distinguished by colors which are set in the `dot_colors` input argument and by their corresponding text legends which are specified in the legends input argument. The name of the resulting graph is set by the `figure_title` input argument.

4.1.2.9 skyplot.m

- **Input arguments:**
  `jsn` (Struct).
- **Output:** Two figures.
- **Description:**
  It uses the `results` struct obtained with the `process_results.m` script to graphically represent the visible satellites positions relative to the user by means of a polar plot which has been customized to resemble a typical polar plot. Each satellite is represented with a filled circle with its own PRN code number (Note: Galileo PRNs are prefixed with an "E"), user 2D position is the origin of the polar plot. It is set to represent the polar plot of the first and last executions of a dataset, with the objective to visualize the orbital trajectory of the visible satellites.

4.2 Validation tests

4.2.1 SNAP-G service validation

For the SNAP-G service to be validated, a test codenamed SNAP.G01 must be passed. Its objective is to validate the operation of the different components that form SNAP-G. This includes the sensor prototypes design, the SNApp software package and the "User scripts" software package as well as other components of the service such as the SNAP API. SNAP.G01 test consists in, firstly, remotely configuring the sensor and secondly, the execution of 50 position

requests. Both configuration and position requests are performed by means of the "User Scripts" software package. Sensor shall automatically get these position requests and perform signal captures accordingly. Then, sensor must send these signal capture files to the platform. The resulting position estimations must have an horizontal error of less than 100 meters for at least 80% of the executions.

In terms of sensor-side, GNSS and Hybrid sensors operation are exactly the same; as it is explained in section 2.1.4, the cellular component of the position estimation is performed by means of a software simulation in the platform. Hybrid sensors must capture and send GNSS samples as they do for SNAP-G service.

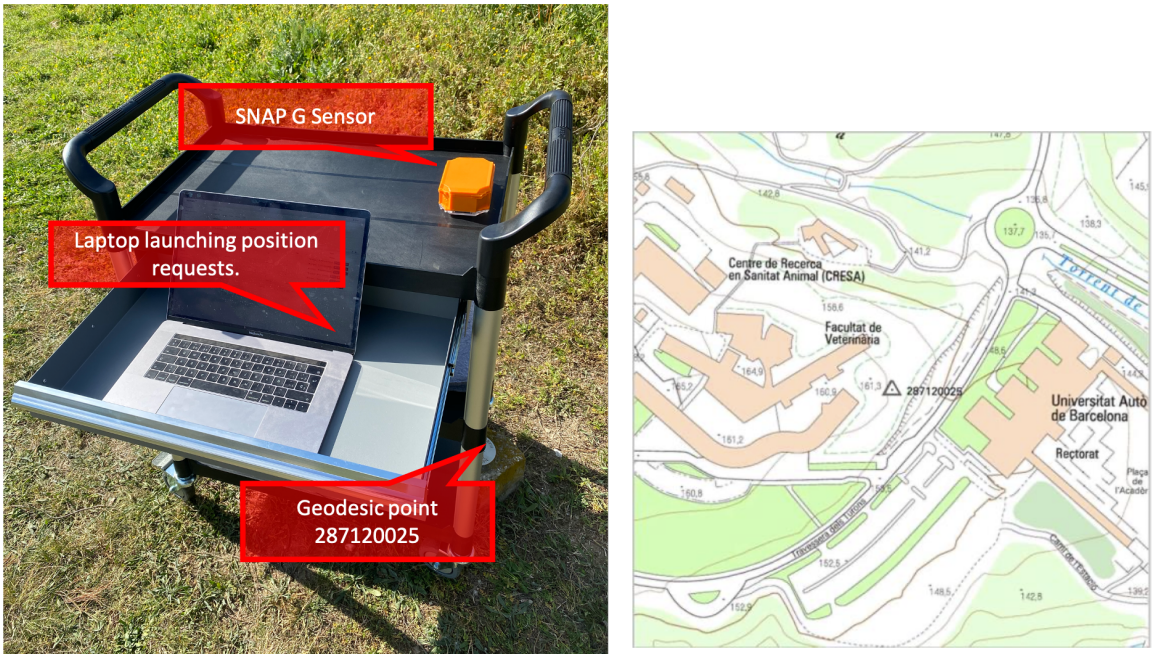
4.2.1.1 Prerequisites

Some requirements must be fulfilled in advance:

- Open-sky visibility should be available to maximize the number of visible satellites and thus, to maximize the chances of having a successful position fix
- Strong Wi-Fi signal must be present in the location the sensor is going to be placed. "eduroam" network is preferred, alternatively a Wi-Fi access point with known credentials can be used.
- Sensors use Network Time Protocol (NTP) for time tagging and processing purposes, so it is important to ensure network port 123 (UDP/NTP) to be open in order to have time errors down to few milliseconds.
- User executing the test is registered and validated in the HANSEL Testbed environment.
- SNApp software must be installed into a SNAP SDR sensor.
- SNAP SDR sensor must be registered in the environment as a G-type sensor.
- Sensor position must be precisely known as it will be used to compute the position error for the estimations and hence, to evaluate the sensor performance.

4.2.1.2 Test description

The sensor was placed in the UAB campus, above the geodesic point 287120025, its location can be seen in figure 4.7(b). The visibility in this point can be fairly considered open-sky, since the two buildings around the geodesic point are relatively far and not tall enough to block substantial sky visibility. The setup used can be also seen in an image 4.7(a).



(a) Setup for SNAP.G01 test

(b) Location for SNAP.G01. Geodesic point 287120025 location.

Figure 4.7: SNAP.G01 test setup and location.

Once the sensor has been calibrated and it is running the listening routine, it is configured with the parameters listed in the table 4.1 using the ”User scripts” software package. After the sensor has been correctly configured, 50 executions are periodically launched with the configuration listed in figure 4.2.

Parameter	Value
Sampling frequency	2.8 MHz
Bandwidth	2.8MHz
Sample Format	IQ
Encoding	INT
Quantization bits	8
File pointer offset	10 ms
Length of the signal to be captured	100 ms
Time between update_request updates	1 s

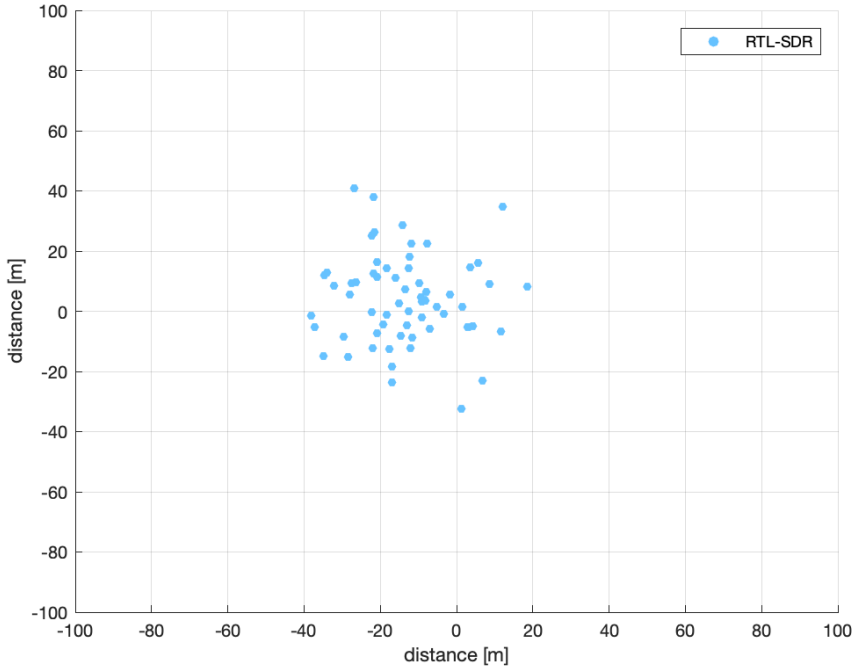
Table 4.1: Sensor configuration parameters for test SNAP.G01

Parameter	Value
GNSS band	1
GNSS system	GPS
Coherent integration time	20ms
Number of non-coherent integrations	1
Enable manual satellite search	false
Enable assisted Doppler frequency	true
Enable automatic generation of assistance data	true
Reference position in latitude, longitude, height	41.50407542222, 2.09924063005, 159.673 (geodesic point 287120025)
Enable interference detection	false

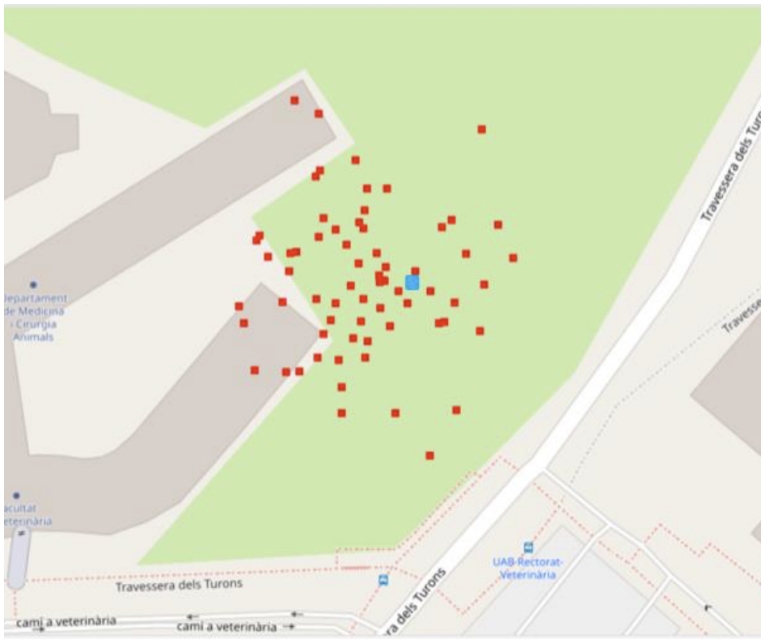
Table 4.2: Configuration parameters for test SNAP.G01.

4.2.1.3 Test results

50 out of 50 executions have been correctly processed. Once this process is finished, the results are retrieved and processed using the "testing tools". FBS visualization service is also used. In figure 4.8 the position estimations are represented using the testing tools (figure 4.8(a)) and the FBS visualization service (figure 4.8(b)). In figure 4.8(b) the sensor reference position is highlighted in blue while in figure 4.8(a) it is located in the origin of the figure. In order to quantify the error position, cumulative distribution function is used to show the probability of obtaining a certain margin of horizontal position error (altitude is not considered), as it can be seen, in figure 4.9, the probability of obtaining an horizontal position error of less than 100 meters is 95.24%. with this test, the workflow involving the GNSS/Hybrid sensors design, the SNApp software package and the "User scripts" software package, is validated, as well as the rest of the rest of involved platform components. It can be concluded that the service requirements REQ-SNAPG-01, REQ-SNAPG-03 and REQ-SNAPG-04 have been fulfilled. As for the sensor software package side, from REQ-SNApp-01 to REQ-SNApp-05 (all of them) have been fulfilled.



(a) Computed GNSS positions with respect to reference position. (Testing tools)



(b) Computed GNSS positions with respect to reference position. (FBS)

Figure 4.8: SNAP.G01 test position estimation results.

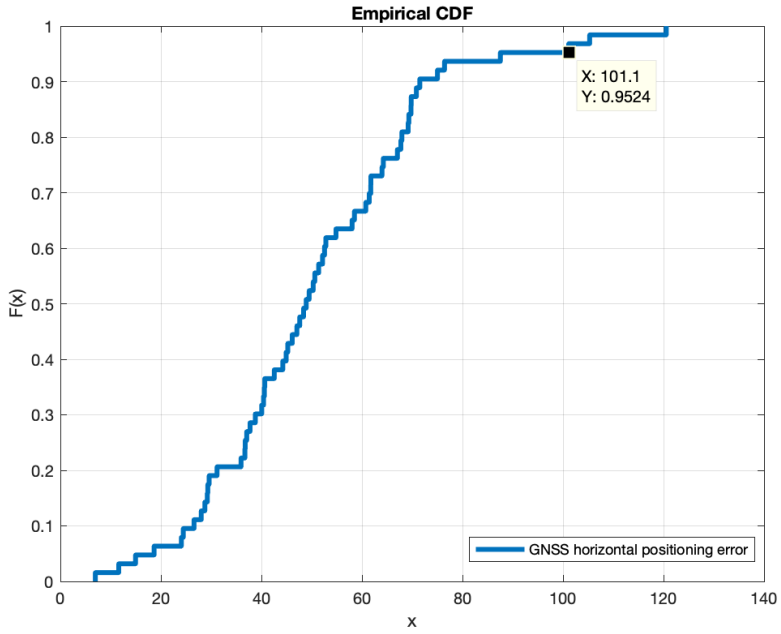


Figure 4.9: Cumulative distribution function of the horizontal position errors obtained in test SNAP.G01

4.2.2 SNAP-C service validation

For the SNAP-C service to be validated, a test codenamed SNAP.C01 must be passed. Its objective is to validate the operation of the different components that form SNAP-C. This includes the sensor prototype design, the SNApp software package and the ”User scripts” software package as well as other components of the service such as the SNAP API. SNAP.C01 test consists in, firstly, remotely configuring the sensor and secondly, the execution of an ”observables” request. Both configuration and observables requests are performed by means of the ”User Scripts” software package, sensor shall automatically get the observables request and perform a signal capture accordingly. Then, sensor must send the resulting signal file to the platform for its processing. The results must determine that at least a single cellular base station has been detected which means that the signal capture have been performed correctly.

4.2.2.1 Prerequisites

Some requirements must be fulfilled in advance:

- Sensor must be placed in a 4G coverage area. Concretely, at least a 4G base station must be within 1km of the sensor.

- Strong Wi-Fi signal must be present in the location the sensor is going to be placed. "eduroam" network is preferred, alternatively a Wi-Fi access point with known credentials can be used.

- User executing the test is registered and validated in the HANSEL Testbed environment.

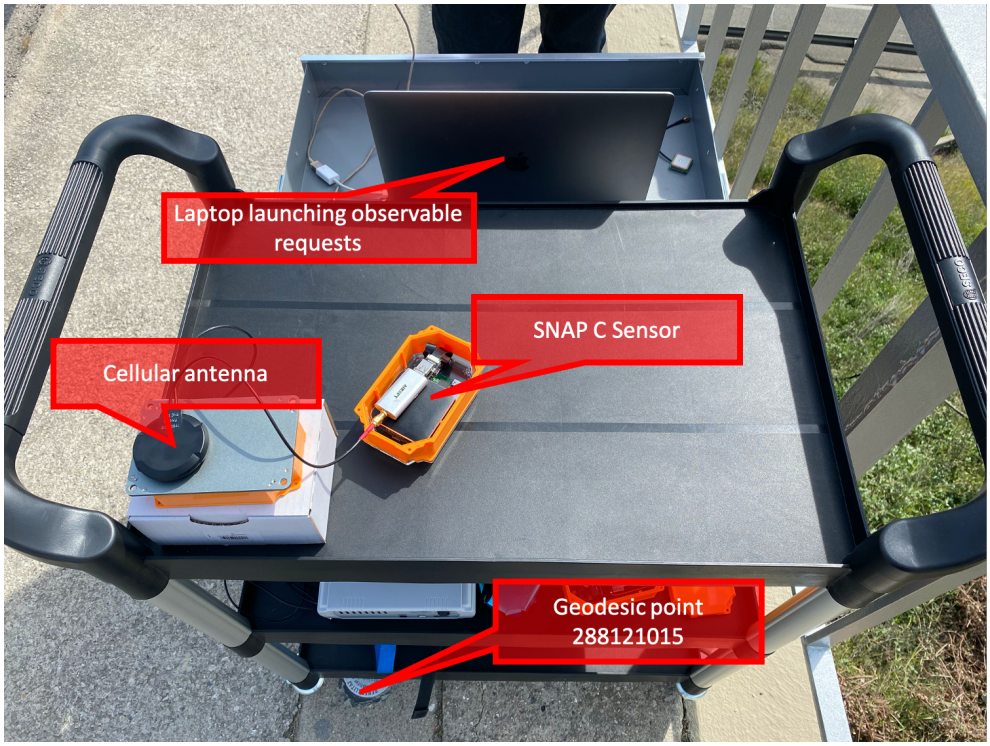
- SNApp software must be installed into a SNAP SDR sensor.

- SNAP SDR sensor must be registered in the environment as a C-type sensor.

4.2.2.2 Test description

Sensor was placed in the geodetic point 288121015, connected to a Cellular antenna, in a setup that can be seen in the next image. As it is stated in the test requirements, the SDR sensor must be surrounded by at least one 4G LTE cellular base station within a radius of one km from the sensor, so at least one base station can be properly detected. This was ensured by means of the website www.cellmapper.net, which shows the deployed base stations in an interactive map. It was found that in fact a base station is within a radius of one km from the sensor, as can be seen in the 4.10(a). Sensor was placed in the geodetic point 288121015, connected to a Cellular antenna, in a setup that can be seen in the figure 4.10(b).

(a) Location for SNAP.C01 test. Geodesic point 288121015.



(b) Setup for SNAP.C01 test.

Figure 4.10: SNAP.C01 test setup and location.

Once the sensor is running the listening routine, it is configured with the parameters listed in the table 4.3 using the ”User scripts” software package. After the sensor has been correctly configured, a single C-type execution is launched from the platform. The configuration of the execution can be observed in table 4.4.

Parameter	**Value**
Sampling frequency	3 MHz
Sample format	IQ
Encoding	INT
Quantization bits	16 bits
Time between /update_request updates	1 second

Table 4.3: Sensor configuration parameters used for test SNAP.C01 test.

Parameter	**Value**
Bandwidth	1.4 Mhz

Table 4.4: Execution configuration parameter(s) used for test SNAP.C01 test.

4.2.2.3 Test results

Once the C-Execution was finished, the SNAP-C service software was able to successfully detect a base station from the signal snapshot. The test is then declared passed. Tthe workflow involving the Cellular sensors design, the SNApp software package and the ”User scripts” software package, is validated, as well as the rest of involved platform components. It can be concluded that requirements REQ-SNAPC-01, and REQ-SNAPG-05 have been fulfilled.

Chapter 5

Experimentation

Once reached this point, the SNAP-G service was already end-to-end validated. SDR sensors, SNAP API, SNAP DB and the CloudGNSSRx external service reliable operation and performance was achieved. Taking advantage of that together with the functionalities the last version of the results processing tools brought, several experiments were made to visualize and quantify the performance of the system for some different configurations. In the following sections, three of the mentioned experiments are explained. First one is a comparison between the performance of the RTL and Airspy Mini sensors using GPS and Galileo constellations separately, second experiment is a comparison between the performance of the system when using compressed and non-compressed signal files and its impact in the sensors power consumption. Finally, the last experiment is a study of the performance of the system when different integration times are used to process GPS signal captures.

5.1 Experiment #1: RTL-SDR vs AirspyMini performance

The objective of this experiment is to analyze the performance obtained when processing GNSS samples captured with the two versions of GNSS SDR sensors which incorporate the RF RTL-SDR and the Airspy Mini RF front ends, respectively. The prerequisites are the same as the ones stated in SNAP.G01 test, listed in section 4.2.1.1.

5.1.1 Experiment description

The study consists in 50 signal captures in the L1 band on the central frequency of 1575.42MHz where the GPS L1 and Galileo E1 signals are found. signal snapshots have a length of 150 ms for both the GPS and Galileo cases. These are sent by the sensor to the cloudGNNSrx platform where they are processed and the user's position is obtained for each capture. The experiment

setup, which is depicted in figure 5.1, consists in two sensors located at a distance of one meter from each other, stationary.

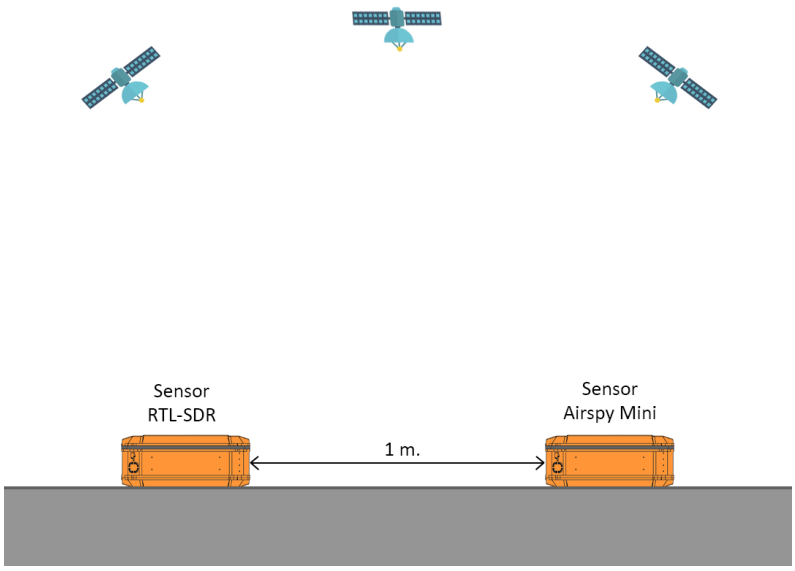


Figure 5.1: SDR performance experiment setup

In both cases, MSB compression is applied to the samples sent to the platform, but they were configured to capture signal with their maximum available capture bandwidth. The configuration of both sensors are listed in the table 5.1 and the configuration for the GPS and Galileo executions is listed in the table 5.2.

Parameter	**RTL-SDR Sensor**	**AirSpyMini Sensor**
Sampling frequency	2.8 MHz	6.0 MHz
Bandwidth	2.8 MHz	6.0 MHz
Sample Format	IQ	IQ
Encoding	SIGN	SIGN
Quantization bits	1	1
File pointer offset	10 ms	10 ms
Length of the signal to be captured	150 ms	150 ms

Table 5.1: Sensor configurations for the experiment.

Parameter	GPS	Galileo
GNSS band	1	1
GNSS system	GPS	Galileo
Coherent integration time	20ms	100ms
Number of non-coherent integrations	5	1
Enable manual satellite search	false	false
Enable assisted Doppler frequency	true	true
Enable automatic generation of assistance data	true	true

Table 5.2: Execution configuration for executions performed using GPS and Galileo constellations.

5.1.2 Experiment results

The results obtained for the case of GPS and Galileo are explained and can be visualized in the following sections. In both cases, some figures are shown; in order to graphically represent quality of the received signal, the ratio between carrier power and noise spectral density (C/N_0) of every satellite acquired is represented by means of a bar graph, in every C/N_0 bar, an error bar representing the interval composed by the maximum and minimum value obtained is added. Next to the signal quality representation, a figure representing the position of the satellites of the GNSS constellation used at the moment of the experiment is shown. Below, a comparison of the error positions obtained by means of a scatter plot can be found and, as a complement for the latter figure, the cumulative distribution function of the position errors obtained is represented.

5.1.2.1 Experiment results using GPS constellation

The average C/N_0 values obtained with the two sensor variants are shown in Fig. 5.2 for the 10 acquired satellites. The horizontal axis represents the satellite identifier through its pseudo-random code number (PRN). The blue bars correspond to the data obtained with the Airspy mini and the red ones to the RTL-SDR. Each bar also indicates the range of values obtained.

As it can be observed in figure 5.2, both sensors acquire most visible satellites shown in figure 5.3 reasonably well except for the number 16 which is expected to occur considering that its elevation is very low, almost on the point of losing line of sight with the sensors. Captures with the RTL-SDR result in values of C/N_0 slightly higher than the Airspy Mini, with a difference of up to 5dB even though the gains of the Airspy Mini were practically set to maximum value. Note

that under ideal conditions, for signals captured outdoors with good visibility of the sky, a value of $C/N_0 \sim 45$ (dBHz) would be expected. The C/N_0 values obtained for both sensors oscillate between 35 and 45 dBHz, slightly lower than expected, essentially due to implementation losses (eg 1-bit quantization, limited bandwidth, etc.), but it is acceptable taking in account that only 100 ms of signal are being integrated. For GPS L1 C/A, the receiver in the cloud is configured such that each capture of 150 ms is processed using $N_{\mathrm{I}} = 5$ coherent correlations of $T_{\mathrm{coh}} = 20$ ms each one, which accumulate in a non-coherent way. This configuration can be observed in table 5.2 (GPS column).

The position errors obtained with each sensor are shown in figure 5.4 by means of a scatter plot. As can be seen, most points are included in an 40 by 40 meters dimension origin-centered area. Examining the cdf comparison of the figure 5.5 it can be determined that errors obtained with the Airspy Mini are around 6.5 meters (1-σ) while for the RTL-SDR they are around 11.5 meters (1-σ). The advantage of the Airspy Mini, at similar values of C/N_0, is given by its greater bandwidth compared to the RTL-SDR.

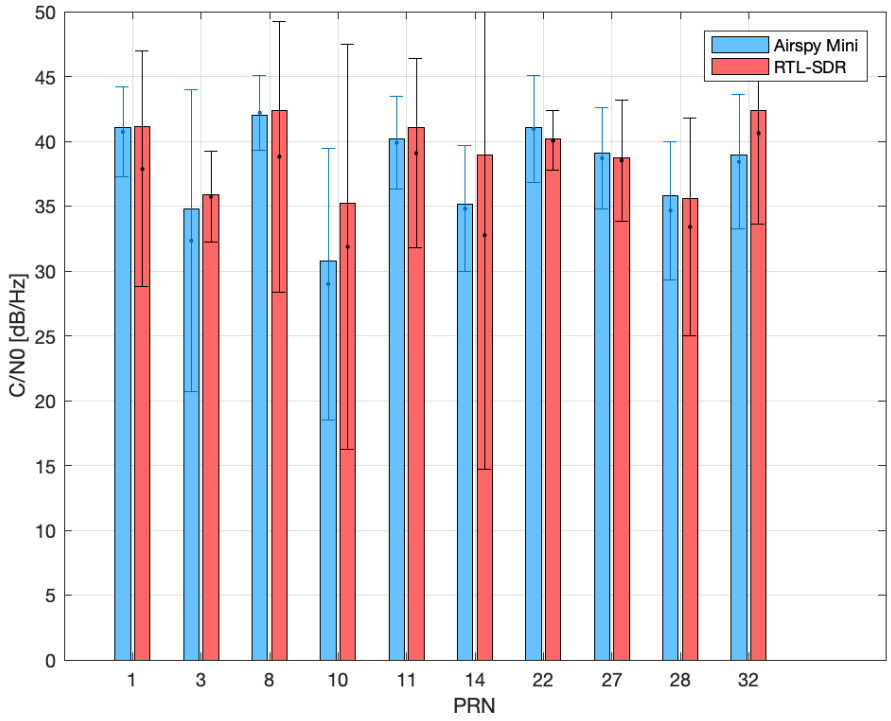


Figure 5.2: Signal level received for each GPS satellite acquired by the platform using both AirSpy Mini and RTL-SDR sensors.

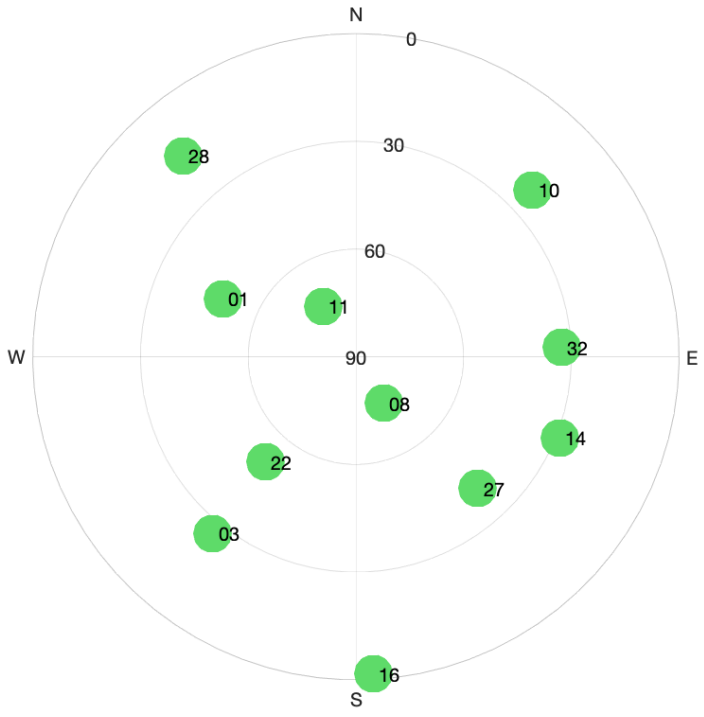


Figure 5.3: Sky plot of GPS constellation at the moment of the experiment.

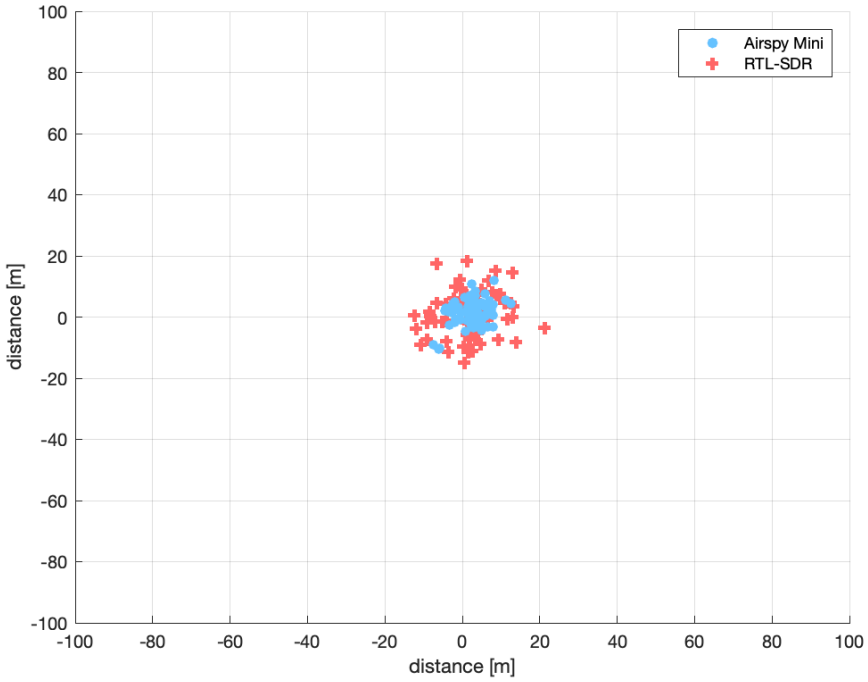


Figure 5.4: Position errors for GPS L1 C/A obtained by the platform using both AirSpy Mini and RTL-SDR sensors.

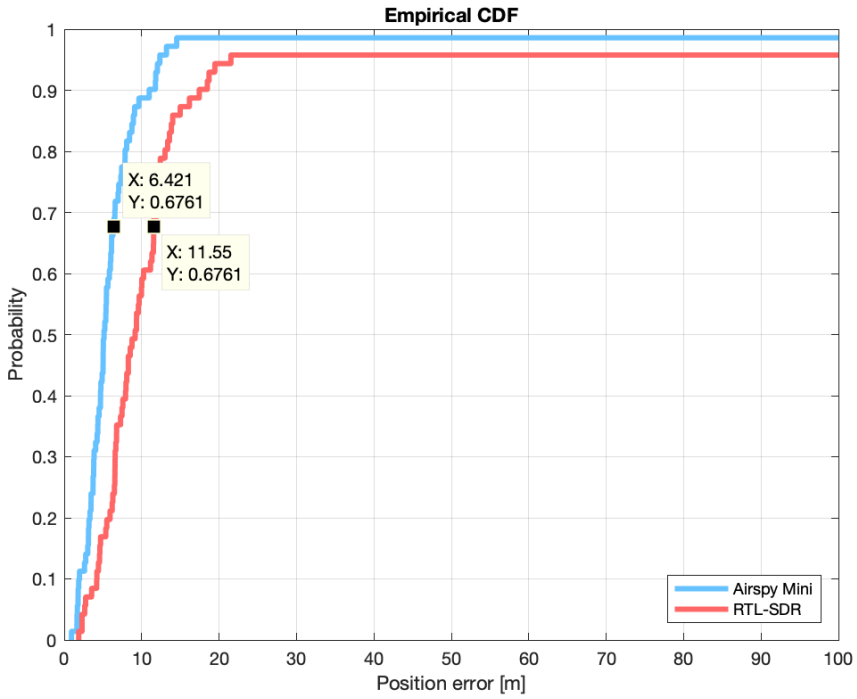


Figure 5.5: Cumulative distribution function of the position errors obtained by the platform using both AirSpy Mini and RTL-SDR sensors. (Using GPS L1 C/A)

5.1.2.2 Experiment results using Galileo constellation

The average values of C/N_0 when processing Galileo E1C are shown in Fig. 5.6 for the 10 acquired satellites, 9 in the case of RTL-SDR, out of the total of 11 shown in figue 5.7. As it can be seen, PRN E22 is not acquired by any of the two sensors, this is because PRN E22 was removed from active service on 8 December 2017 until further notice for constellation management purposes. Captures with Airspy Mini result in higher values than RTL-SDR. This happened due to the fact that the bandwidth of the RTL-SDR is less than the necessary to properly process the Galileo E1C signal (minimum $\sim$ 4 MHz). As a result, the spectrum at the output of the RTL-SDR is greatly shortened, causing power losses and signal distortion. Therefore the results of the RTL-SDR with Galileo are expected to be much worse than with the Airspy Mini. This fact can be clearly seen in figure 5.8.

For Galileo E1C, the 100 ms of the capture is processed totally in a coherent way due to the absence of bits of information in this signal (it is a pilot signal), with which $T_{\text{coh}} = 100$ ms, and therefore $N_{\text{I}} = 1$, as it can be seen in table 5.2.

Examining the cumulative distribution function comparison of the figure 5.9, it can be seen that the position errors obtained with the Airspy Mini sensors are around 15 meters (1-σ), somewhat worse than those obtained with the same head for the case of GPS L1 C/A. Although the visibility conditions of the GPS satellites were better at the time of capture (more satellites on the zenith and with better geometry), the main degradation is due to the antenna, whose bandwidth is more intended for GPS signals than for Galileo ones. This makes the antenna the bottleneck of the entire chain, causing the spectrum of the Galileo signal to be clipped and distorted from origin, with a significant degradation as observed when comparing the blue bars in figures 5.2 and 5.6. This effect is amplified in the case of the sensor with RTL-SDR, whose results for Galileo deteriorate up to 50 meters (1-σ).

In short, due to the nature of GPS L1 signals, an RF head with $\sim$ 2 MHz bandwidth like the RTL-SDR is enough to obtain user positions with acceptable error. However, in the case of Galileo E1, an antenna and a front-end with at least twice the bandwidth is necessary.

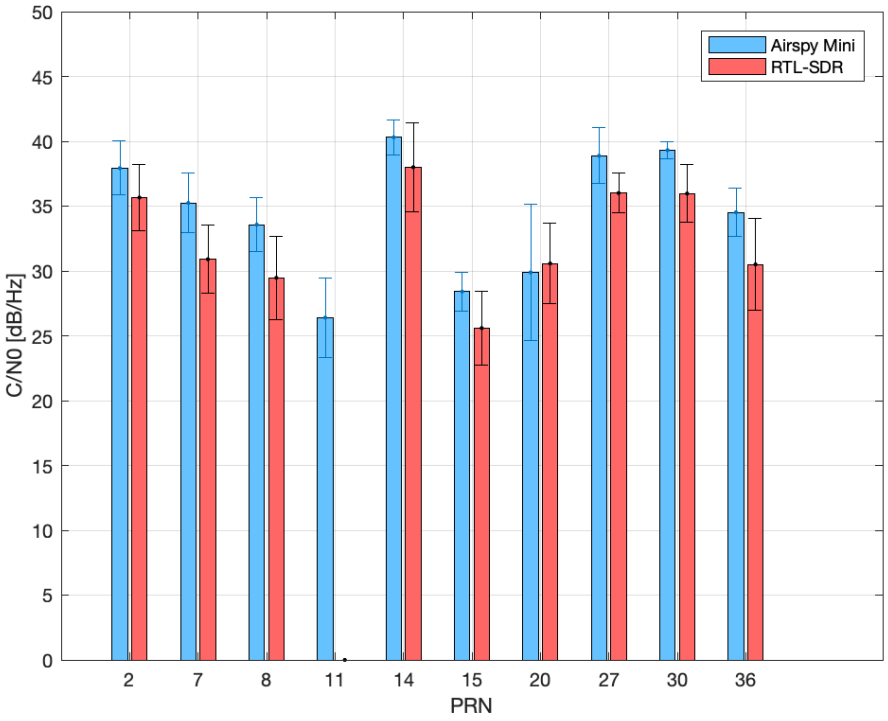


Figure 5.6: Signal level received for each Galileo satellite acquired by the platform using both AirSpy Mini and RTL-SDR sensors.

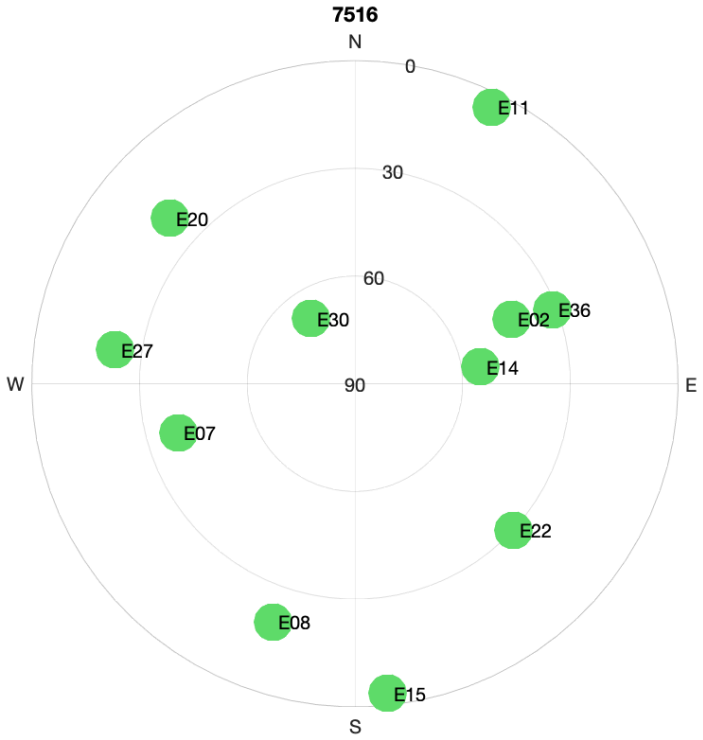


Figure 5.7: Sky plot of Galileo constellation at the moment of the experiment.

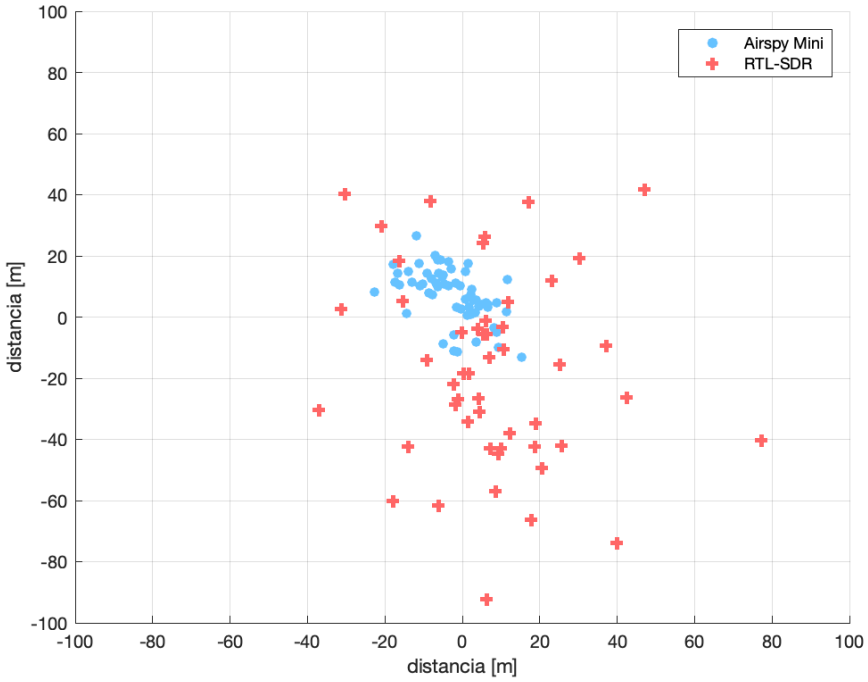


Figure 5.8: Position errors for Galileo E1C obtained by the platform using both AirSpy Mini and RTL-SDR sensors.

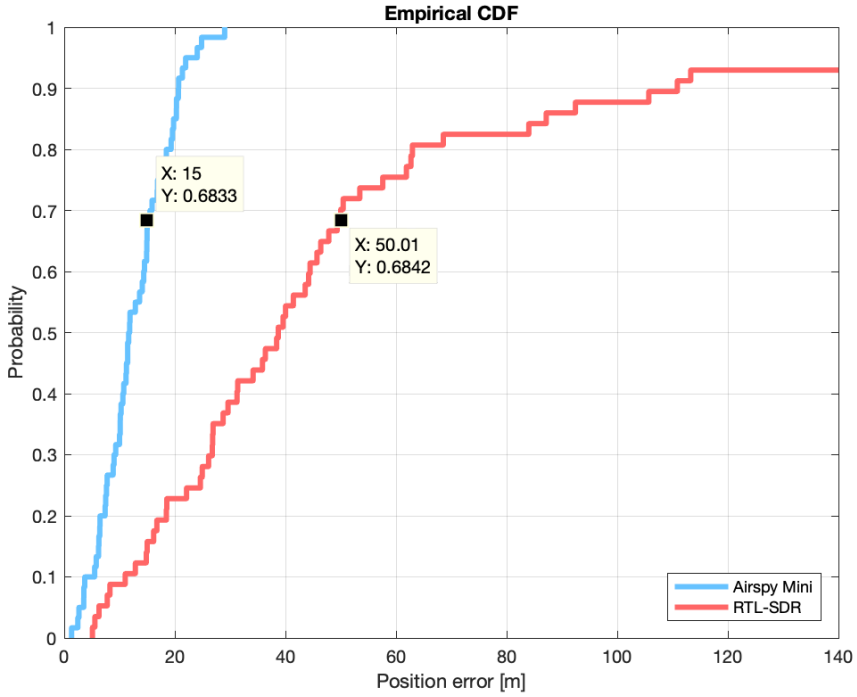


Figure 5.9: Cumulative distribution function of the position errors obtained by the platform using both AirSpy Mini and RTL-SDR sensors. (Using Galileo E1C)

5.2 Experiment #2: Raw samples compression

As it is explained in previous sections, the developed sensors have the ability of compressing signal snapshots reducing the output signal file by a factor of 8 in the case of the RTL-SDR and 16 in the case of the Airspy mini. The advantage of performing this compression is that, this way, files which sensors send to the server are much smaller and as a consequence, they spend less power transmitting them via Wi-Fi. In order to quantify the power savings caused by the files compression, the power consumption of a sensor in three operation states were measured using a simple power discharge measuring device. These three operation states are listed below:

- **Listening:** Sensor is running the listening routine, without performing any signal capture.
- **Capturing with compression:** Sensor is running the listening routine periodically capturing in L1 band 100 ms signal snapshots and sending them to the platform, at maximum executions per minute rate. Compression is applied before sending the file to the platform.
- **Capturing with compression:** Sensor is running the listening routine periodically capturing in L1 band 100 ms signal snapshots and sending them to the platform, at maximum executions per minute rate. No compression is applied.

In figure 5.10 the results of the measurements is depicted by means of a bar graph. It can be seen that sending the data files to the platform with no compression applied implies a power consumption increment of 46.78% when performing captures at maximum rate. Note that in figure 5.10 only the data for the case of an Airspy Mini sensor is shown, the measurements were also performed win an RTL-SDR sensor and results were similar.

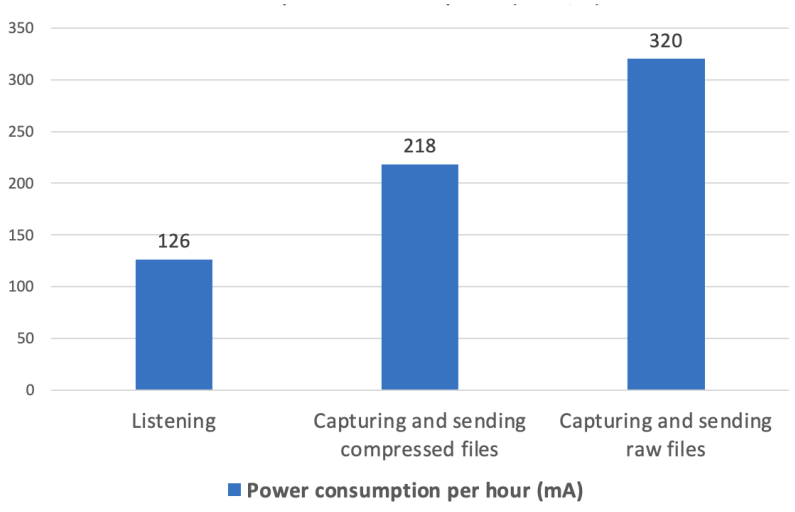


Figure 5.10: Sensor power consumption graph. (AirSpy Mini sensor).

It is clear that the difference in terms of power consumption is significant, the objective of this study is to also quantify the differences between processing compressed and non-compressed signal snapshots in terms of received signal quality and position error. The prerequisites for this experiment are the same as the ones stated in SNAP.G01 test, listed in section 4.2.1.1.

5.2.1 Experiment description

The study consists in 50 signal captures in the L1 band on the central frequency of 1575.42MHz where the GPS L1 and Galileo E1 signals are found, performed y two sensors: one using an RTL-SDR and another using an Airspy Mini with the sensor configuration listed in table 5.3 and the execution configuration listed in the "Galileo" row of table 5.2. The location and distribution of the sensors is exactly the same as in figure 5.1 and the sky plot at the moment of the experiment is depicted in figure 5.11. Once the 50 signal captures were performed, the output files were:

1. Processed with no modifications.
2. Compressed and reprocessed.

Hence, the same sample files have been used for compression and non-compression processing.

Parameter	**RTL-SDR Sensor**	**AirSpyMini Sensor**
Sampling frequency	2.8 MHz	6.0 MHz
Bandwidth	2.8MHz	6.0 MHz
Sample Format	IQ	IQ
Encoding	INT	INT
Quantization bits	8	16
File pointer offset	10 ms	10 ms
Length of the signal to be captured	150 ms	150 ms

Table 5.3: Sensor configurations for the experiment.

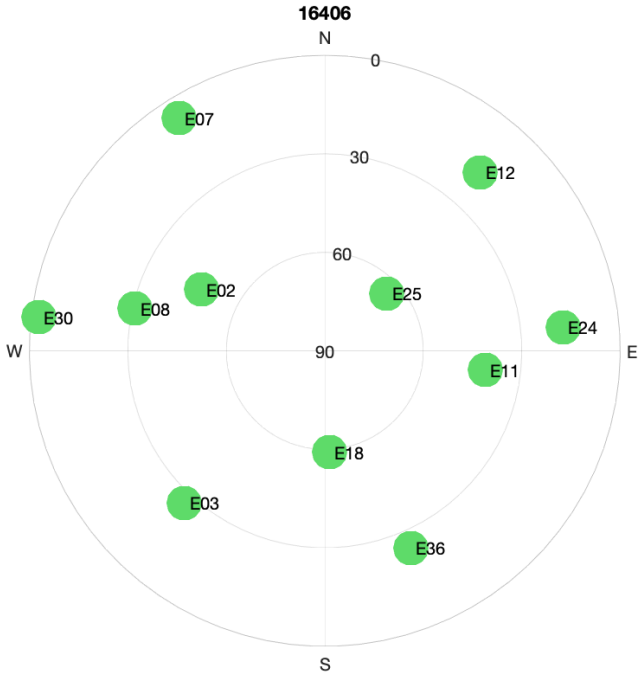


Figure 5.11: sky plot of Galileo constellation at the moment of the experiment

5.2.2 Experiment results

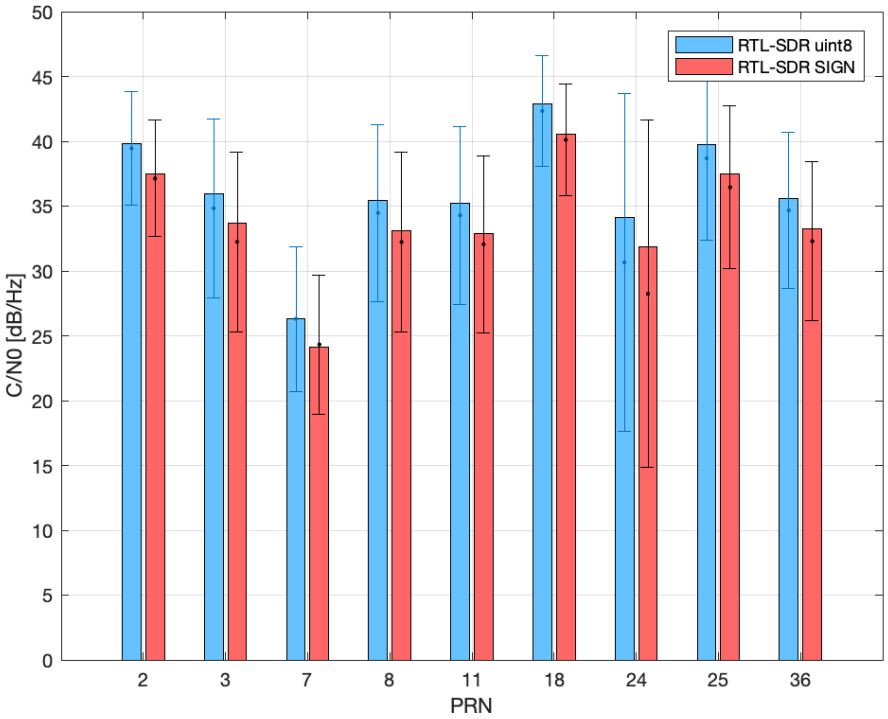
The results obtained for both SDR front-ends are explained below. For both cases, some figures are shown. In order to graphically represent quality of the received signal, the ratio between carrier power and noise spectral density (C/N_0) of every acquired satellite is represented by means of a bar graph, in every C/N_0 bar, an error bar representing the interval between by the maximum and minimum value obtained is added. Below, a comparison of the error positions obtained by means of a scatter plot can be found and lastly, as a complement for the latter figures, the respective cumulative distribution functions of the position errors obtained are represented.

In figure 5.12(a), a comparison between the C/N_0 values acquired by the RTL-SDR sensor with and without compression is depicted and the same is shown for the Airspy Mini sensor in figure 5.12(b). For the case of the RTL-SDR it can be clearly seen that compression causes a constant C/N_0 loss of $\sim 2.5dB$ for every acquired satellite, the same phenomenon can be observed for the Airspy Mini, where the C/N_0 constant loss is slightly greater with a value of $\sim 4dB$.

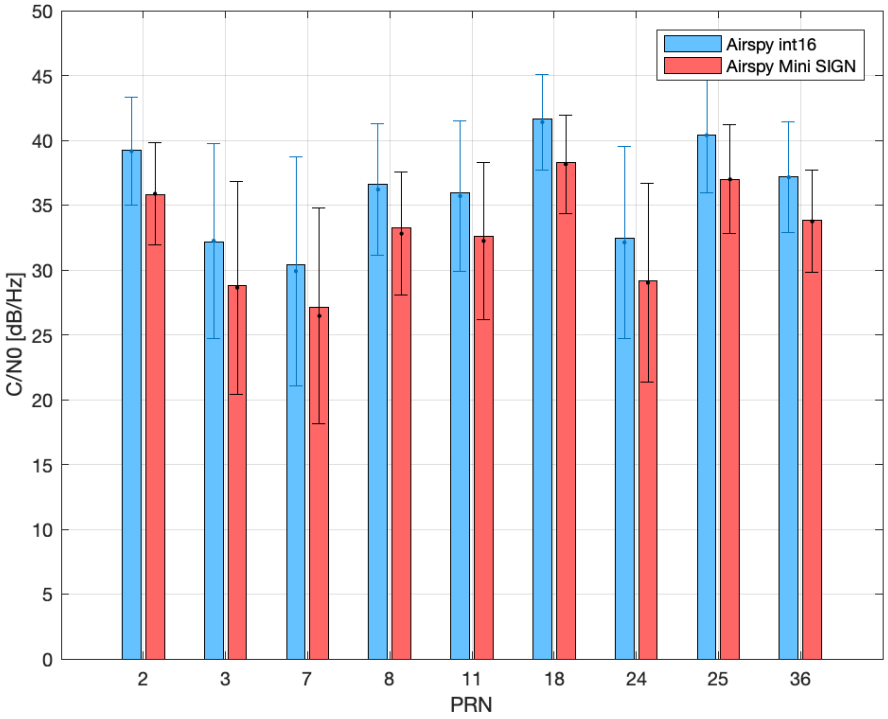
In figure 5.13(a), a scatter plot comparison between the position errors obtained by the platform using the RTL-SDR sensor with and without compression applied is depicted and its respective cumulative distribution function is shown in figure 5.14(a), the same figures can be observed for the Airspy Mini sensor in 5.13(b) and in 5.14(b) respectively. As it can be

seen in both position error scatter plots, despite the fact of the C/N_0 degradation caused by compression, there is no noticeable difference in terms of position error. This can be confirmed observing the cumulative distribution function comparisons, where traces follow almost the same path.

In conclusion, for applications with power-consumption restrictions, signal data compression before transmission could be worth the implementation.

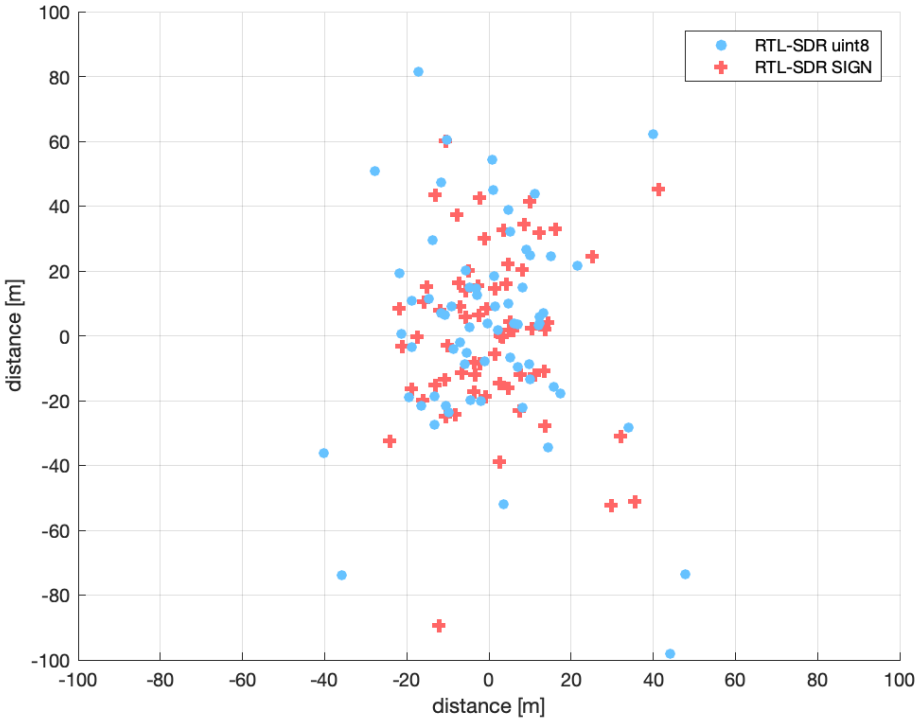


(a) Signal level received for each Galileo satellite acquired by the platform comparison using RTL-SDR captured snapshots without compression applied (uint8) and with 8:1 compression applied (SIGN).

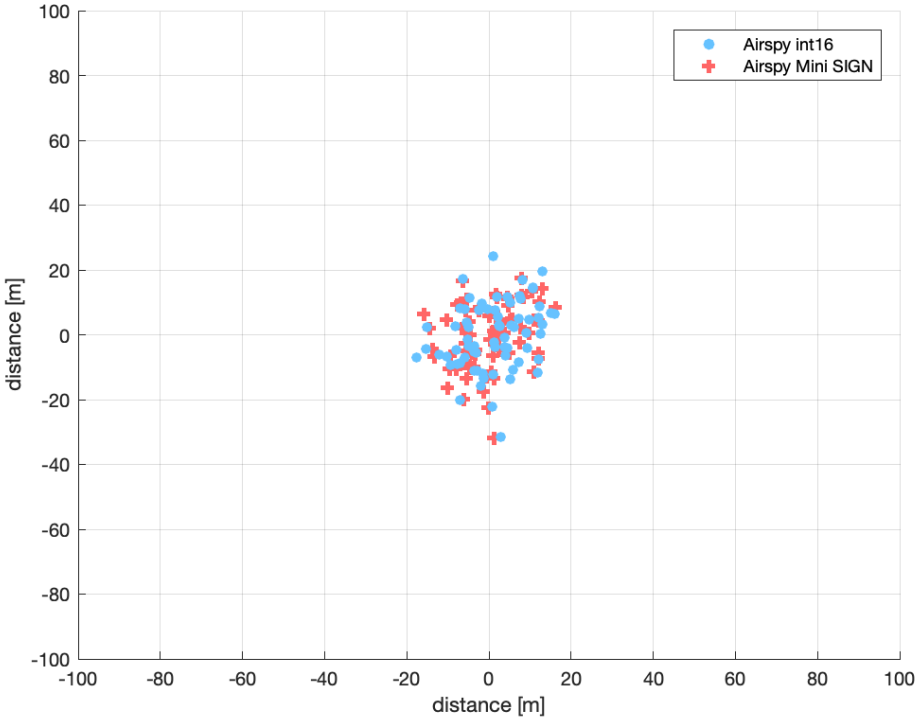


(b) Signal level received for each Galileo satellite acquired by the platform comparison using Airspy Mini captured snapshots without compression applied (int16) and with 16:1 compression applied (SIGN).

Figure 5.12: Signal level received comparison using data compression and raw signal files.

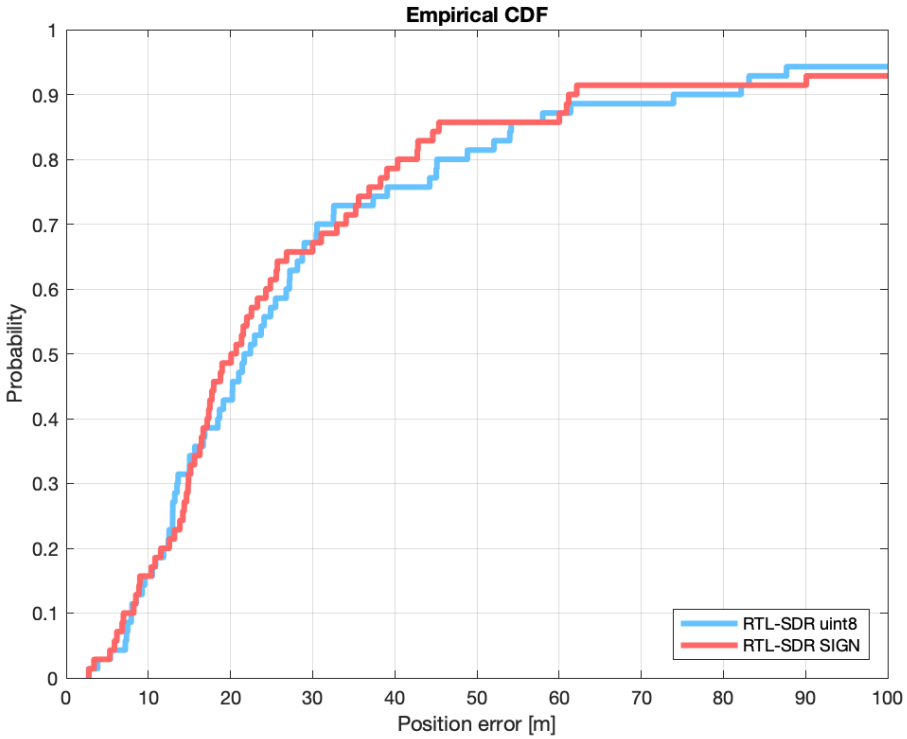


(a) Position errors for Galileo E1C obtained by the platform comparison using RTL-SDR captured snapshots without compression applied (uint8) and with 8:1 compression applied (SIGN)

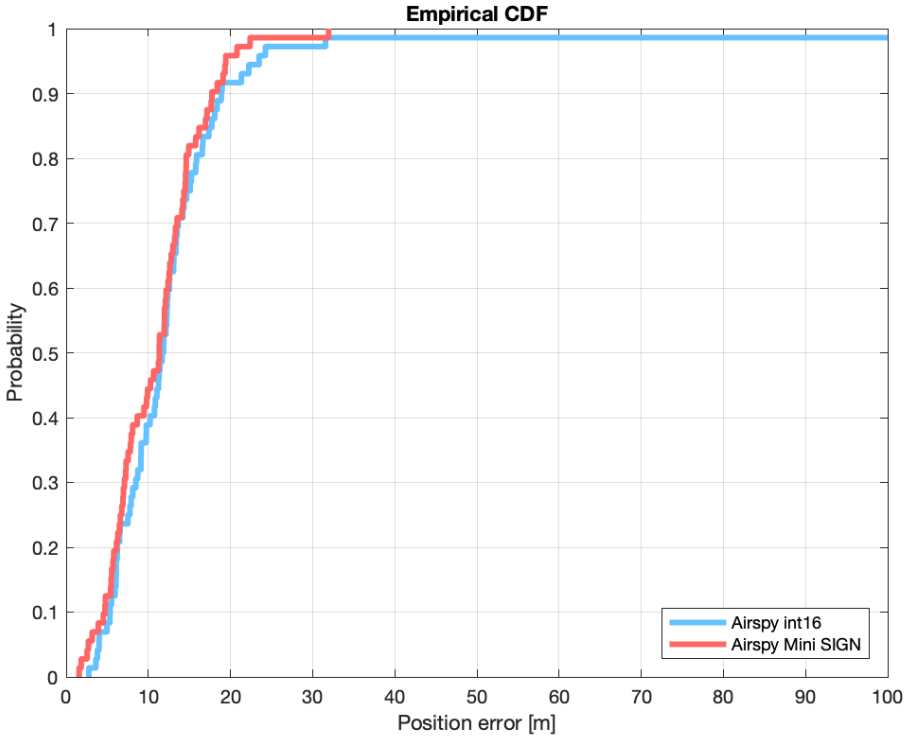


(b) Position errors for Galileo E1C obtained by the platform comparison using Airspy Mini captured snapshots without compression applied (int16) and with 16:1 compression applied (SIGN)

Figure 5.13: Position errors comparison using data compression and raw signal files.



(a) cumulative distribution function of the position errors obtained by the platform comparison using RTL-SDR captured snapshots without compression applied (uint8) and with 8:1 compression applied (SIGN)



(b) cumulative distribution function of the position errors obtained by the platform using Airspy Mini captured snapshots without compression applied (int16) and with 16:1 compression applied (SIGN)

Figure 5.14: Position error CDF comparison using data compression and raw signal files.

5.3 Experiment #3: Performance as a function of the integration time

As it is explained in previous sections, the platform can be configured to process signal snapshots with a specified correlation time. The more correlation time, the more time will take to the cloud receiver to compute the position fix, on the other hand, an increase of the correlation time should improve the position estimation accuracy.

5.3.1 Experiment description

The study consists in 50 GPS-signal captures in the L1 band on the central frequency of 1575.42MHz, performed by two sensors: one using an RTL-SDR and another using an Airspy Mini with the sensor configuration listed in table 5.4. The location and distribution of the sensors is exactly the same as in figure 5.1 and the prerequisites for this experiment are the same as the ones stated in SNAP.G01 test, listed in section 4.2.1.1. Once the 50 signal captures were performed, the output files were:

1. Processed with 100 ms of integration time.
2. Reprocessed with 20 ms of integration time.

Hence, the same files have been used for both 20 ms and 100 ms of integration time processing.

Parameter	**RTL-SDR Sensor**	**AirSpyMini Sensor**
Sampling frequency	2.8 MHz	6.0 MHz
Bandwidth	2.8MHz	6.0 MHz
Sample Format	IQ	IQ
Encoding	SIGN	SIGN
Quantization bits	1	1
File pointer offset	10 ms	10 ms
Length of the signal to be captured	150 ms	150 ms

Table 5.4: Sensor configurations for the experiment.

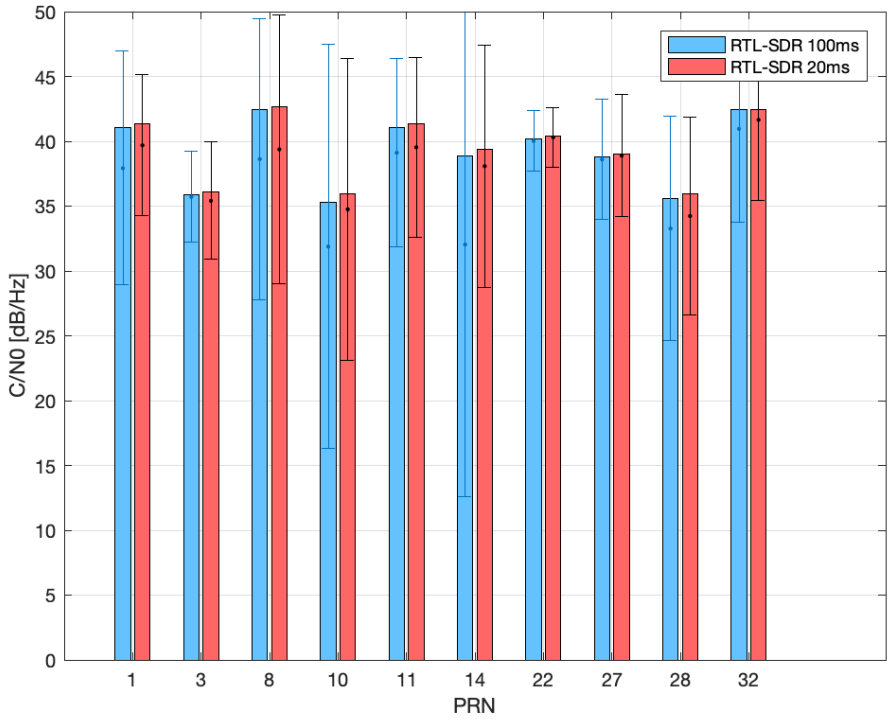
5.3.2 Experiment results

The results obtained for both SDR front-ends are shown below. In both cases, some figures are shown. In order to graphically represent quality of the received signal, the ratio between carrier power and noise spectral density (C/N_0) of every GPS satellite acquired is represented by

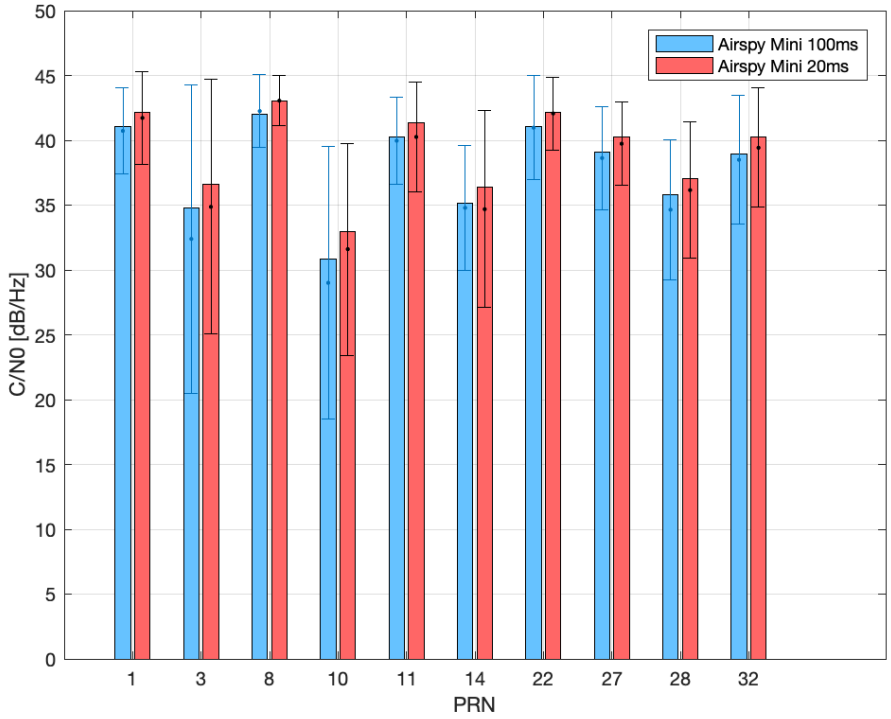
means of a bar graph, in every C/N_0 bar, an error bar representing the interval composed by the maximum and minimum value obtained is added. Below, a comparison of the error positions obtained by means of a scatter plot can be found and lastly, as a complement for the latter figures, the respective density functions of the position errors obtained are represented.

In figure 5.15(a), a comparison between the C/N_0 values acquired by the RTL-SDR sensor with 20 ms and 100 ms of integration time is depicted and the same is shown for the Airspy Mini sensor in figure 5.15(b). For the case of the RTL-SDR it can be seen that the increment of integration time does not cause any improvement in the C/N_0 values, while for the case of the Airspy Mini, against the odds, executions with only 20 ms of integration time acquired a higher value of C/N_0 in every PRN with an increase of up to $\sim 2.5dB$.

In figure 5.16(a), a scatter plot comparison between the position errors obtained by the platform using the RTL-SDR sensor using 100 ms and 20 ms of correlation time is depicted and its respective cumulative distribution function is shown in figure 5.17(a), the same figures can be observed for the Airspy Mini sensor in figures 5.16(b) and in 5.17(b) respectively. As it can be seen in figures 5.16(a) and 5.17(a), in the case of the RTL-SDR there is not significant improvement in terms of position accuracy but in the case of the Airspy Mini, there is a noticeable increase in position accuracy of almost 4 meters or $+38,23\%$ $(1-\sigma)$, this can be observed in figure 5.16(b) and can be confirmed in figure 5.17(b).
In conclusion, since the cloud receiver takes around twice the time to process the executions with 100 ms of integration time regarding ones with only 20 ms, it may seem that there is no logical reason to use 100 milliseconds or more since there are not significant improvements in the case of the RTL-SDR and in the case of the Airspy Mini only 4 meters of accuracy improvement may not be significant for most applications this approach would be used in, but there are several scenarios where using longer integration times could be helpful, for instance, when intermittent interference is present.

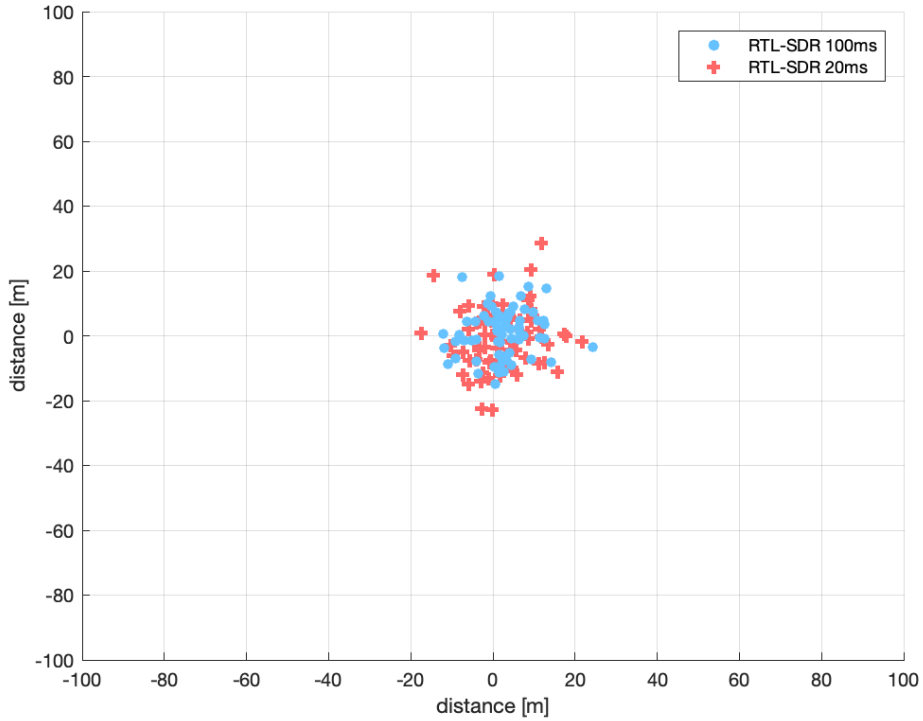


(a) Signal level received for each GPS satellite acquired by the platform comparison using RTL-SDR with 20 ms and 100 ms of integration time

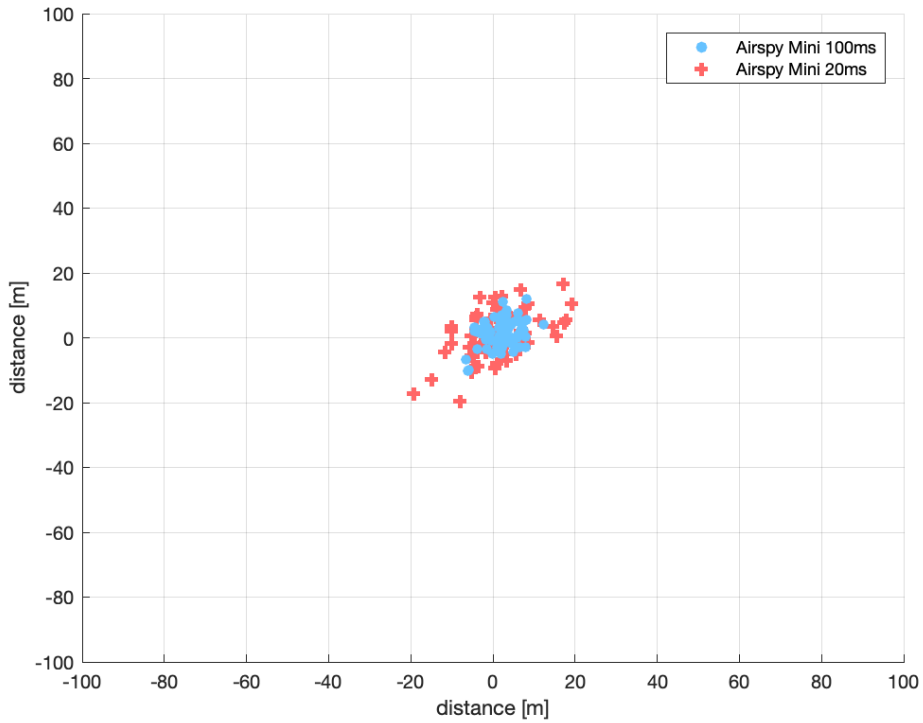


(b) Signal level received for each GPS satellite acquired by the platform comparison using Airspy Mini with 20 ms and 100 ms of integration time

Figure 5.15: Signal level received comparison using 100 ms and 20 ms of integration time

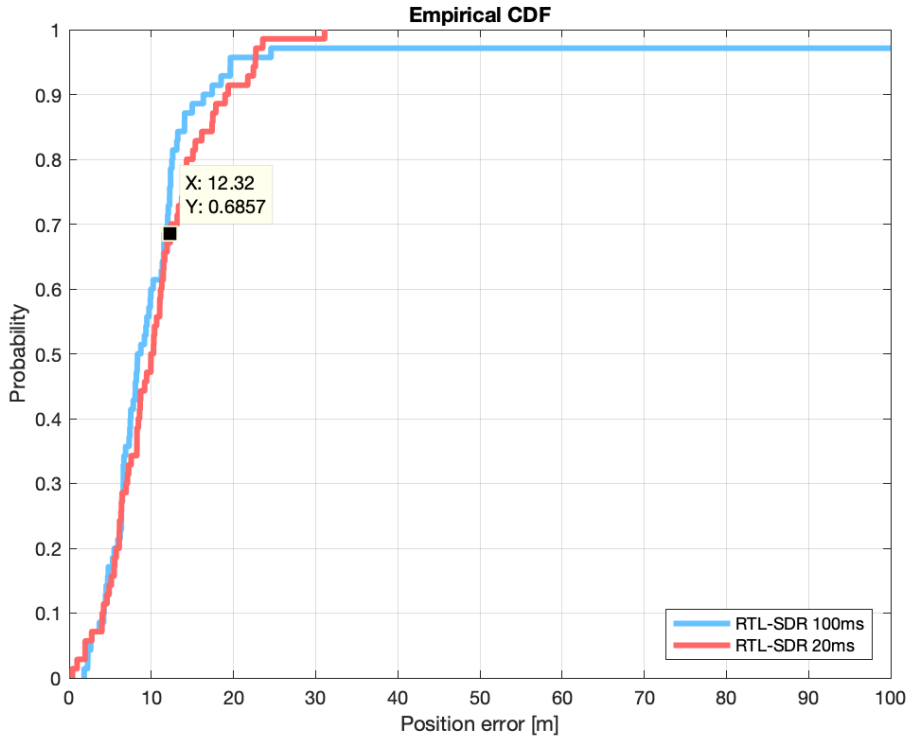


(a) Position errors for GPS L1 C/A obtained by the platform comparison using RTL-SDR with 20 ms and 100 ms of integration time

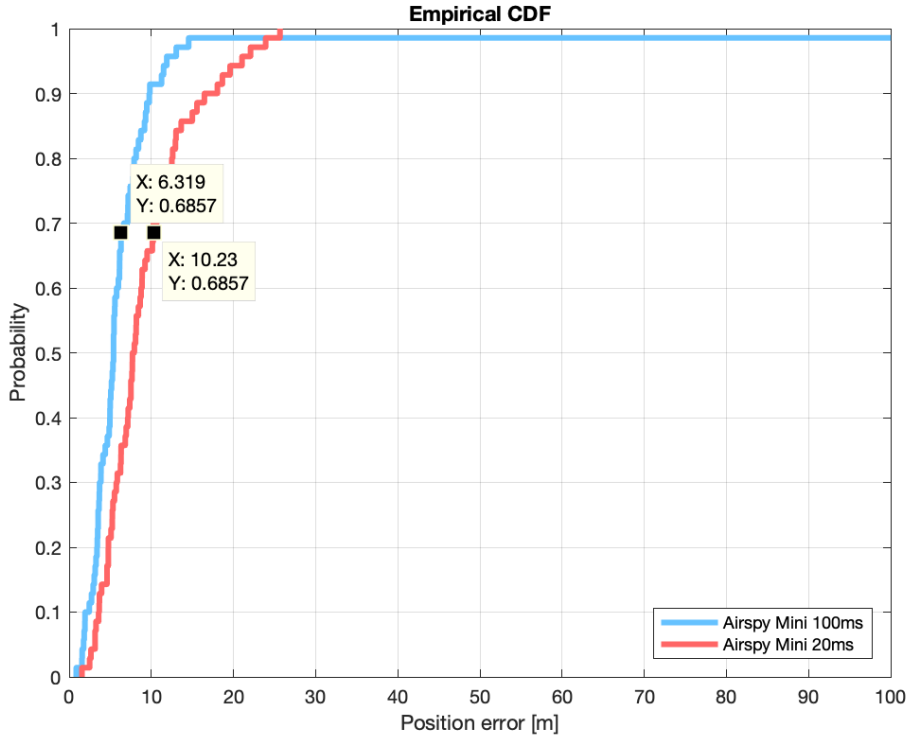


(b) Position errors for GPS L1 C/A obtained by the platform comparison using Airspy Mini with 20 ms and 100 ms of integration time

Figure 5.16: Position errors comparison using 100 ms and 20 ms of integration time



(a) cumulative distribution function of the position errors obtained by the platform comparison using RTL-SDR with 20 ms and 100 ms of integration time



(b) cumulative distribution function of the position errors obtained by the platform comparison using Airspy Mini with 20 ms and 100 ms of integration time

Figure 5.17: Position error CDF comparison using 100 ms and 20 ms of integration time

Chapter 6

Conclusions and future lines of work

After the process of realization of this thesis, which has lasted more than a year, some conclusions can be extracted:

1. It have been demonstrated that the cloud computing approach for GNSS and Cellular samples processing is feasible. A low cost sensor prototype has been validated whose concept has an enormous potential for future applications in Internet of Things and Smart Cities fields.

2. The initial sensor software requirements have been pushed further creating a software package which can be shared by different kind of sensors with different hardware components, this together with the use of disk images cloning conformed a system of sensors generation which consists basically in a single mouse click.

3. Similar to the previous point, the same happened with the sensor operation which was also pushed further by the addition of the startup routine and the reset/power on button, thanks to that, sensor operagion incredibly changed from the initial stages where every sensor had to be constantly and manually monitored, the operation with multiple devices was unthinkable. At the end of the project, sensors needed zero monitoring, and the only manual interaction they needed was literally pressing the power on button.

4. Related to the previous two points, the same happened in terms of performance. Once the required perfomance was achieved, it was decided not to stop there and keep iterating to improve it as much as possible. As a result, the performance achieved at the final project stages were several times better then the required one.

5. A software package for sensor operators (users) was developed which generated an independence from UAB-external services. This way the full control of the sensors development was kept from the beginning.

6. A software package for the processing of large datasets of execution results were developed. This fact was a key for the sensor development since this toolkit was widely used during the sensor development and testing phases. Furthermore, thanks to this toolkit some experiments about the sensors performance were made which generated very interesting data which was used to publish a paper [1] in the Seminario Anual de Automática, Electrónica Industrial e Instrumentación (SAAEI) on September 2020.

As of future lines of work there are still some pending tasks and ideas that would be interesting to develop:

1. Due to the covid-19 pandemic, the final review of the platform is still pending since May 2020. The intention is to perform the sensor installation physically in the European Space Research and Technology Centre installations (ESTEC) in Noordwijk, The Netherlands, once the global situation allows.

2. As for the sensor, an idea worth to develop is a second much powerful sensor with even broader bandwidth using the HackRF One [21] front-end and the Raspberry Pi 4 board.

3. Some additional features could be added to the results processing tools such as the representation of the C/N_0 values relation with the elevation of satellites.

4. SDR calibration by GNSS satellites dopplers proved to be very effective but it had to be done manually. The automatizing of this process would increase even more the sensor's performance and operation autonomy.

5. A dual SDR sensor could be developed that would bring some possibilities such as sensors with a GNSS and a Cellular antenna. This way GNSS, Cellular and Hybrid sensors would share te exact same software and hardware. Furthermore, dual band GNSS positioning techniques could be implemented as well as array processing techniques.

6. Taking in account the current performance and of the platform and the previous points, countless interesting experiments and studies could be performed.

6.1 Publications

The results obtained in the experiment covered in section 5.1 were of special relevance in the development of the project. Hence, it was decided to write a paper for its the 27th session of "Seminario Anual de Automática, Electrónica Industial e Instrumentación" (SAAEI), which was selected for its presentation at the seminar conference, held on September 2, 2020. The article is titled "Remote Processing of GNSS Signals Using Low Cost RF Sensors" [1], it describes the

capabilities of the GNSS sensor, the SNAP API and the CloudGNSSRx as well as the mentioned experiment realization and results.

Bibliography

[1] J. Gáñez-Fernández, A. Pérez-Conesa, G. Seco-Granados, J. A. López-Salcedo,"Remote Processing of GNSS Signals Using Low Cost RF Sensors", *Annual Seminar on Automation, Industrial Electronics and Instrumentation (SAAEI)*, Ciudad Real (Spain), 2020.

[2] A. Minetto, F. Dovis, A. Vesco, M. Garcia-Fernandez, À. López-Cruces, J. Luis-Trigo, J. Trigo, A. Pérez-Conesa, J. Gáñez-Fernández, G. Seco-Granados, J. A. López-Salcedo,"A testbed for GNSS-based Positioning and Navigation Technologies in Smart Cities: The HANSEL Project", *MDPI Smart Positioning and Timing*, 2020.

[3] Stewart, R. and others,"A low-cost desktop software defined radio design environment using MATLAB, Simulink, and the RTL-SDR", *IEEE Communications Magazine Vol. 53, pages 64-71*, 2015.

[4] Cass, S.,"A $40 software defined radio", *in IEEE Spectrum Vol. 3, pages 22-23*, 2013.

[5] Ulversoy, Tore,"Software defined radio: challenges and opportunities", *IEEE Communications Surveys & Tutorials Vol. 12, pages 531-550*, 2010.

[6] Harris, Fred and Lowdermilk, Wade, "Software defined radio: Part 22 in a series of tutorials on instrumentation and measurement", *IEEE Instrumentation & Measurement Magazine Vol. 13, pages 23-32*, 2010.

[7] Galileo Signal plan [Online],
available: https://gssc.esa.int/navipedia/index.php/Galileo_Signal_Plan.

[8] European GNSS (Galileo) Open Service, Signal-in-space interface control document. [Online] available: https://www.gsc-europa.eu/sites/default/files/sites/all/files/Galileo-OS-SIS-ICD.pdf

[9] Galileo PRS [Online], available: https://gsa.europa.eu/security/prs.

[10] Galileo Navigation message [Online],
available: https://gssc.esa.int/navipedia/index.php/Galileo_Navigation_Message

[11] About the Cirocomm 580R [Online],
available: http://www.cirocomm.com/en-global/products_ciro/detail/GBA-803G

[12] About he Rubicon by Open h [Online],
available: http://openh.io/rubicon/

[13] About the RTL-SDRv3 [Online],
available: https://www.rtl-sdr.com/about-rtl-sdr

[14] About the AirSpy Mini [Online],
available: https://airspy.com/airspy-mini

[15] Celestrak, NORAD Two-Line Element Sets [Online],
available: https://www.celestrak.com/NORAD/elements

[16] Simplified General Perturbation model orbital simulator (SGP4) [Online],
available: https://pypi.org/project/sgp4/

[17] CloudGNSSrx platform [Online],
available: http://cloudGNSSrx.com

[18] kalibrate-rtl [Online],
available: https://github.com/steve-m/kalibrate-rtl

[19] Putty [Online],
available: https://putty.org

[20] About SDR# [Online],
available: https://www.rtl-sdr.com/tag/sdrsharp/

[21] About HackRF One [Online],
https://greatscottgadgets.com/hackrf/one/