Centre Tecnològic de Telecomunicacions de Catalunya

WORKSHOP Software defined receivers for satellite navigation

Instructor Guide

TELECO RENTA

PLAN DE PROMOCIÓN DE LOS ESTUDIOS DE TELECOMUNICACIÓN

Financiado por la Unión Europea NextGenerationEU

Plan de Recuperación, Transformación y Resiliencia

UNICO I+D

GOBIERNO DE ESPAÑA

This workshop has been prepared by the Centre Tecnològic de Telecomunicacions de Catalunya (CTTC), within the context the project "Plan de promoción de los estudios de Telecomunicación" (UNICO 5G I+D program).

The content of this workshop, including the source code, the instructor manual and the student manual are made available for anyone interested.

Contact: telecorenta@cttc.cat

Authors: Carles Fernández, Monica Navarro.

Document history

Revision	Date	Affected sections	Modifications
V1.0	10/10/2024	All	First edition

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

OUTLINE

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

List of Figures

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

Acronyms & abbreviations

DLL	Delay Locked Loop
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
SDR	Software Defined Radio
SW	Software
PLL	Phase Locked Loop
PSD	Power Spectral Density
RF	Radio Frequency
UDP	User Datagram Protocol
USRP	Universal Software Radio Peripheral

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

1. Workshop objectives

This workshop aims to introduce the students to the theory, implementation and use of GNSS receivers. It particularly, it focuses on the software implementation of GNSS receivers.

The workshop will also demonstrate the real-time operation of a SDR (Software Defined Radio) GNSS receiver implementation for two different modes: either working with real-time signals, received during the workshop execution, or working with pre-recorded raw GNSS signal files.

For the first option the student workshop will require to have available several RTL-SDR devices and the corresponding GPS antennas. Preferably, one device for each student.

The workshop is intended to be organized as follows:

1) For the instructor to first introduce GNSS navigation and positioning principles, in lecture mode or as he/she finds more suitable. The introductory material (section 2 from the Student Guide) is also contained in the student guide to allow the student post-consultation and review. It is accounted for 30 min introduction to the theory and fundamentals. This guide also includes the documentation details of the GNSS-SDR Signal Processing Blocks (section 3) developed by CTTC, which is the developer and maintainer of GNSS-SDR open-source software. The content of section 3 is for advanced students with interests to dwell into SDR development. It may also provide the instructor with additional material if he/she wants to elaborate on some of the signal processing blocks, such as synchronization, etc.

Once the introduction part is completed the actual practical work for the students shall start by:

2) The students to go through the practical aspects of downloading and setting up the necessary software and configuration to run a SDR receiver. We provide a container-based solution in order to speed up the process. Nevertheless, the installation part (section 2 in the Student Guide) may take 15'-20' for all students to have the set-up ready.

   The installation considers the option of having Windows OS and Linux OS. There is also a note if the SW receiver is intended to be use with a raspberry.

3) The actual exercise of running the receiver and getting the actual position from the satellites in view. The post-processing FILE version may be faster to execute.

   If the hardware devices are available, the instructor may want to allow the students to test the receiver in different set-ups: static, moving around the campus (once the installation is completed), etc.

4) The visualization of the results.

Summary of workshop Objectives:

- Intuitive Understanding: Gain an intuitive understanding of how Global Navigation Satellite Systems shape our world and underpin our daily experiences.
- Receiver Insights: Develop a comprehensive grasp of how GNSS receivers calculate your precise position and time, demystifying the technology within your devices.
- Hands-On Experience: Dive into the world of software-defined GNSS receivers, creating your experiments and configuring your own receiver to obtain position and time data directly from satellites.

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

The details for the installation of the required software and configuration files are detailed in the Student Guide.

The specification of the selected hardware devices is also included in the Student Guide. Although the installation details and workshop exercises have been validated with the described hardware, the workshop may be adapted to other compatible SDR devices.

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

2. Introduction to Global Navigation Satellite System and SDR receivers

Have you ever wondered how your smartphone accurately pinpoints your location on a map? It's all thanks to an incredible technology called Global Navigation Satellite System (GNSS). Let's take a closer look at how it works.

Each GNSS consists of a network of satellites that orbit the Earth, transmitting signals that enable devices like smartphones and other electronic gadgets to determine their precise location anywhere on the planet. The most well-known GNSS is the Global Positioning System (GPS) developed by the United States, but there are others: GLONASS, developed by Russia; Galileo, developed by Europe; and BeiDou, developed by China. Each of these systems comprises approximately 32 satellites orbiting the Earth at an altitude of around 24,000 km from the planet's surface. These satellites complete two full orbits around the Earth in a single day.

Figure 1 - Representation of GNSS satellite constellations (GPS, GLONASS, Galileo, and BeiDou).

GNSS satellites are the modern version of lightphares. Technology has allowed us to place them in the sky instead of the seacoast, for better visibility, and to make them emit radio waves with ingenious waveforms instead of light. In the same way that mariners were able to estimate its 2D position over the sea by looking at two or more lightphares at the same time and the aid of a map, GNSS receivers can compute their 3D position and time by looking at four or more satellites at the same time. The problem that they solve (that is, *knowing where you are*) is in fact quite complex has kept mankind busy since very ancient times.

The art of finding the way from one place to another is called navigation. Until the 20th century, the term referred mainly to guiding ships across the seas. Indeed, the word "navigate" comes from the Latin *navis* (meaning "ship") and *agere* (meaning "to move or direct"). Today, the word also encompasses the guidance of travel on land, in the air, and in inner and outer space.

Navigation has been (and today still *is*) a major scientific and technological challenge. Navigation seems to start very early in time: according to Chinese storytelling, the compass was invented and used in wars during foggy weather before recorded history. Early mariners followed landmarks

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

visible onshore, until other technological breakthroughs came into play: the magnetic compass (c. 13th century), the astrolabe (c. 1484), the sextant (1757), the use of lighthouses and buoys, or the seagoing chronometer (1764). These inventions fueled disciplines such as surveying and geodesy, but also had a dramatic impact in transportation, and thus in economics. Today, the virtuous circle of science, technology, and business around navigation is still spinning and well alive.

The original proposal for a passive one-way ranging system, consisting of 24 satellites with atomic clocks in medium Earth orbits to achieve a 12-hour period, received approval from the U.S. Defense Department in December 1973. Named the "Navstar Global Positioning System," it was initially designed for military purposes. In February 1978, the first Block I developmental Navstar/GPS satellite was launched, followed by three more satellites by the end of that year. Between 1977 and 1979, over 700 tests were conducted with the assistance of Aerospace engineers, confirming the accuracy of the integrated space/control/user system.

In 1983, President Ronald Reagan authorized the use of Navstar (later known as GPS) by civilian commercial airlines to enhance navigation and safety in air travel. This marked the initial step toward authorized civilian usage, leading to the commercial availability of handheld GPS units by 1989. The Magellan NAV 1000, weighing 700 grams, offering limited battery life, and priced at $3,000, was among the first commercially available devices.

GPS technology continued to advance through the 1980s and 1990s, making its debut in cellphones in 1999. In 2000, the U.S. government approved plans to add three additional GPS signals for non-military use, and the "selective availability" program, limiting civilian GPS accuracy, was terminated. Concurrently, Russia, the European Union, and the Republic of China developed their own satellite-based navigation systems: GLONASS (operational since 1993), Galileo (offering Early Operational Capability from December 15, 2016), and BeiDou (operational since December 2011), respectively.

Receiver technology rapidly evolved, featuring smaller size, improved integration with other systems, increased precision, reduced power consumption, and lower costs. Today, GNSS receivers find pervasive use across various domains. These include transportation (land, sea, and air), surveying and mapping, crucial timing and synchronization applications in finance, telecommunications, and scientific research, as well as various scientific fields such as geophysics, meteorology, and seismology. Additionally, GNSS technology finds application in agriculture, military and defence operations, emergency and disaster management, mining and construction, environmental monitoring, and consumer devices like smartphones, cameras, and gaming platforms. From monitoring key infrastructures and tracking tectonic plate movements to enhancing user experiences in electronic games like the popular Pokémon GO, satellite-based navigation technology plays a pivotal role in a diverse array of our daily activities.

This Workshop provides an in-depth exploration of the fascinating realm of GNSS technology. Commercial receivers typically take the form of integrated circuits, commonly known as "chips," housed in small black boxes where the magic of navigation unfolds. While this packaging is convenient for integration into compact devices, it doesn't offer users insight into the internal workings. However, researchers at CTTC have introduced an innovative approach to GNSS receiver implementation: an open-source, software-defined receiver (see http://gnss-sdr.org). This revolutionary concept replaces the specialized integrated circuit with lines of code that can run on a general-purpose microprocessor, such as the one in your computer. This approach empowers users to craft their own GNSS receiver, experiment with tuning parameters, and gain a deeper understanding of the intricacies behind seemingly routine activities, such as checking Google Maps on your smartphone.

Financiado por la Unión Europea NextGenerationEU · Plan de Recuperación, Transformación y Resiliencia · UNICO I+D · GOBIERNO DE ESPAÑA

3.1 Fundamentals of satellite-based navigation systems

Picture each satellite as a unique singer, performing a distinct song. Each song has the same duration and repeats continuously, like a musical loop.

Now, imagine you've memorized all these songs. By hearing a snippet of one, you can instantly calculate how much time has passed since it began. Multiplying this time by the speed at which the song travels from the satellite to your ears, you can determine the distance between the satellite and you. If you also know the precise location of that satellite, you can deduce that you're situated on the surface of a sphere, with the satellite at the centre and the distance as the radius. Exciting, right?

Extend this idea to listening to three different songs at once. You can then draw three spheres, each centred on a different satellite and with varying radii. Your exact position becomes the magical point where all three spheres intersect. Voila! You've cracked the code and found your location.

This analogy simplifies the core principle of Global Navigation Satellite Systems. While in reality, satellites don't sing but emit periodic electromagnetic waves, the fundamental idea remains as straightforward as this musical analogy.

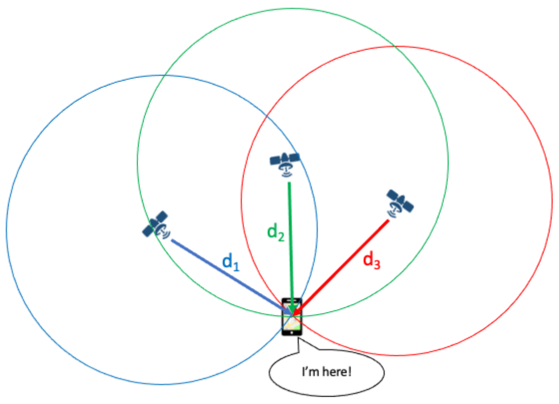


Figure 2 - Oversimplified diagram of how satellite-based navigation systems work

Diving into the intricacies, the analogy mentioned earlier simplifies the process of computing our 3D position based on the distances to three known points. We've explored how these distances can be calculated as delays multiplied by the propagation speed of electromagnetic waves. However, to accurately determine the delay caused by the propagation time from the satellite's antenna to your receiver's antenna (your 'ears' in the analogy), we encounter a crucial requirement – the need for a clock ticking in perfect harmony with the internal clock of the satellite.

Any deviation from synchronization would introduce errors in the delay estimation. Here, precision matters significantly. Given that electromagnetic waves travel at the speed of light, a mere 1-millisecond error can result in an approximate 300-kilometer positional discrepancy! To mitigate this, all GNSS satellites are equipped with extremely precise atomic clocks. These clocks, however, come at a considerable cost.

Yet, the challenge doesn't end there. For an accurate position fix, your receiver would also require an atomic clock to ensure precise timekeeping. Unfortunately, these clocks are exorbitantly expensive, far surpassing the cost of your everyday smartphone. Consequently, relying on atomic clocks in consumer-grade devices is not a practical solution.

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

To understand how we can overcome this hurdle, let's attempt to encapsulate what we've learned in an equation. Consider that we are receiving the signal transmitted by satellite *i*, positioned at ($Xsat_i$, $Ysat_i$, $Zsat_i$), while our receiver is at an unknown location (X_{user}, Y_{user}, Z_{user}). Additionally, the internal clock of our receiver deviates by an unknown amount, $\Delta clock_{user}$ seconds, from the atomic clock of the satellite. We can express this relationship in the following equation:

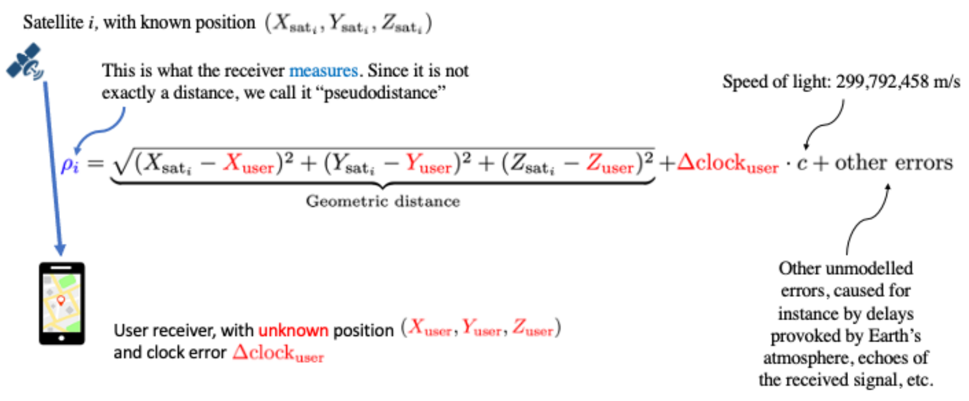

Satellite i, with known position $(X_{\mathrm{sat}_i}, Y_{\mathrm{sat}_i}, Z_{\mathrm{sat}_i})$
This is what the receiver measures. Since it is not exactly a distance, we call it "pseudodistance"
Speed of light: 299,792,458 m/s
$\rho_i = \underbrace{\sqrt{(X_{\mathrm{sat}_i} - X_{\mathrm{user}})^2 + (Y_{\mathrm{sat}_i} - Y_{\mathrm{user}})^2 + (Z_{\mathrm{sat}_i} - Z_{\mathrm{user}})^2}}_{\text{Geometric distance}} + \Delta\mathrm{clock}_{\mathrm{user}} \cdot c + \text{other errors}$
Other unmodelled errors, caused for instance by delays provoked by Earth's atmosphere, echoes of the received signal, etc.
User receiver, with unknown position $(X_{\mathrm{user}}, Y_{\mathrm{user}}, Z_{\mathrm{user}})$ and clock error $\Delta\mathrm{clock}_{\mathrm{user}}$

Figure 3 - Mathematical model for receiver's measurement

This equation reveals that each delay measured from a satellite (called "pseudodistance") introduces four unknown quantities. Notably, these unknowns are consistent across all satellites, as they are only dependent on the location and clock characteristics of the receiver. As you may anticipate, resolving these four unknowns requires incorporating measurements from at least a fourth satellite. Consequently, we can formulate a system of (at least) four equations, each corresponding to a different satellite, collectively constituting a set of simultaneous equations:

$$
\begin{aligned}
\rho_1 &= \underbrace{\sqrt{(X_{\mathrm{sat}_1} - X_{\mathrm{user}})^2 + (Y_{\mathrm{sat}_1} - Y_{\mathrm{user}})^2 + (Z_{\mathrm{sat}_1} - Z_{\mathrm{user}})^2}}_{d_1} + \Delta\mathrm{clock}_{\mathrm{user}} \cdot c + \text{other errors}_1 \\
\rho_2 &= \underbrace{\sqrt{(X_{\mathrm{sat}_2} - X_{\mathrm{user}})^2 + (Y_{\mathrm{sat}_2} - Y_{\mathrm{user}})^2 + (Z_{\mathrm{sat}_2} - Z_{\mathrm{user}})^2}}_{d_2} + \Delta\mathrm{clock}_{\mathrm{user}} \cdot c + \text{other errors}_2 \\
\rho_3 &= \underbrace{\sqrt{(X_{\mathrm{sat}_3} - X_{\mathrm{user}})^2 + (Y_{\mathrm{sat}_3} - Y_{\mathrm{user}})^2 + (Z_{\mathrm{sat}_3} - Z_{\mathrm{user}})^2}}_{d_3} + \Delta\mathrm{clock}_{\mathrm{user}} \cdot c + \text{other errors}_3 \\
\rho_4 &= \underbrace{\sqrt{(X_{\mathrm{sat}_4} - X_{\mathrm{user}})^2 + (Y_{\mathrm{sat}_4} - Y_{\mathrm{user}})^2 + (Z_{\mathrm{sat}_4} - Z_{\mathrm{user}})^2}}_{d_4} + \Delta\mathrm{clock}_{\mathrm{user}} \cdot c + \text{other errors}_4 \\
\vdots &= \qquad \vdots
\end{aligned}
$$

Figure 4 - System of equations for the determination of user's position and time.

In this system, each equation represents the delay measured from a specific satellite. Solving this set of equations simultaneously allows us to determine the precise 3D coordinates of the receiver, and the exact time in which those signals have been received. With atomic clock precision!

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

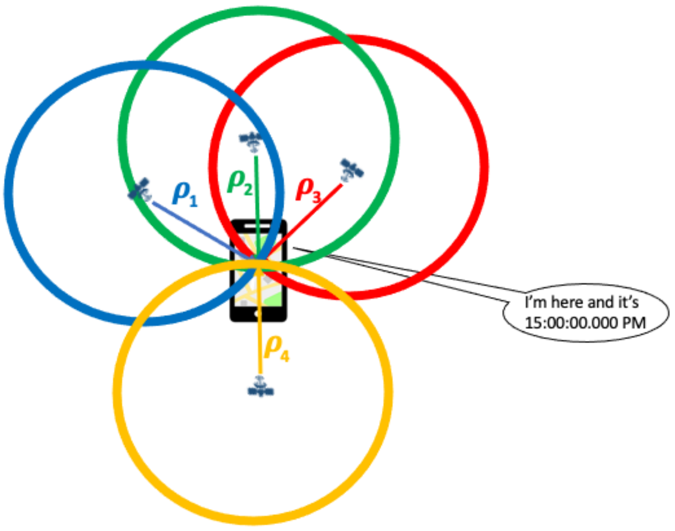


Figure 5 - An improved diagram of how satellite-based navigation systems work.

At this juncture, you might be wondering how the receiver can discern the exact position of each satellite at any given time. Drawing a parallel with our musical analogy, satellites ingeniously alter the lyrics of their 'songs' while steadfastly adhering to the tempo. This alteration involves the transmission of crucial orbital parameters that allow the receiver to compute the satellites' precise positions at any given moment. Amazing, isn't it?

In technical terms, this transmission of orbital parameters is referred to as the *navigation message*. Essentially, it constitutes a set of critical parameters that convey information about the satellite's orbital trajectory, health status, and other pertinent details. It serves as the informational cornerstone, allowing the receiver to calculate the positions of the satellites in real-time and ensuring the accuracy and reliability of the entire Global Navigation Satellite System (GNSS).

In essence, the system of equations above encapsulates the fundamental principle of GNSS navigation. As technology advances, researchers and engineers around the world are continually refining methods for solving it in more accurate and efficient ways. This involves the constant improvement of statistical models for the "other errors" components, such as troposphere and ionosphere characterization, accounting for the potential presence of signal echoes, and the mitigation of electromagnetic interferences, among other factors.

The evolution of GNSS technology relies on the continuous enhancement of these statistical models and digital signal processing methods. Additionally, it involves the creation of innovative technologies that facilitate the practical implementation of these methods. The pursuit of more accurate, reliable, and robust GNSS solutions drives collaboration among experts in various fields, ensuring that the technology not only keeps pace with the demands of today but also anticipates the challenges of tomorrow.

Crucially, this collaborative effort spans beyond mathematicians, physicists, or telecommunication / electrical / computer engineers. It extends to the entire community of users who seek to harness this technological marvel for leisure, business, or exploratory endeavours. From everyday commuters relying on navigation apps to global enterprises optimizing logistics, GNSS has become an indispensable tool, and the ongoing collaboration between developers and users ensures its adaptability and effectiveness in meeting the diverse needs of a global user base.

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

3.2 How the signals transmitted by GNSS satellites are?

Up to this point, we've grasped the significance of receiving signals from a minimum of four GNSS satellites simultaneously to compute our position and time accurately. Let's delve deeper into how these signals are characterized.

Two primary approaches are employed for signal description: the time domain and the frequency domain. Although mathematically equivalent, they offer distinct perspectives on the features of the signal.

In the time domain, signals are described through their waveform, a representation moulded by a process known as *modulation*. This transformative process turns a binary stream of 0s and 1s into voltage variations. For instance, positive voltage may be assigned to a bit 1, and negative voltage to a bit 0. In GNSS signals, this includes the combination of their uniquely distinctive code (their "song" in our musical analogy) and the navigation message containing info about their orbit and status (the "lyrics"), so the receiver can compute its exact location. Those two streams of 0s and 1s, each one changing their values at a different rate, are the multiplied by a sinusoid, which acts as the carrier frequency that allocates the signal's energy in the desired frequency band. The resulting product is then transmitted by the satellite's antenna. This process is sketched in Figure 6.

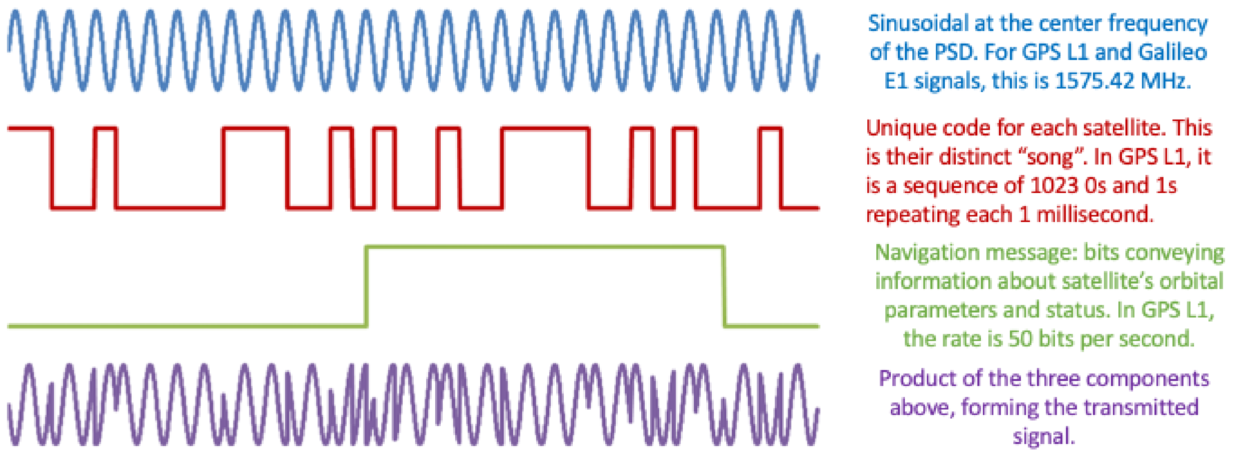

Sinusoidal at the center frequency of the PSD. For GPS L1 and Galileo E1 signals, this is 1575.42 MHz.
Unique code for each satellite. This is their distinct "song". In GPS L1, it is a sequence of 1023 0s and 1s repeating each 1 millisecond.
Navigation message: bits conveying information about satellite's orbital parameters and status. In GPS L1, the rate is 50 bits per second.
Product of the three components above, forming the transmitted signal.

Figure 6 - Signal transmitted by a GNSS satellite expressed in the time domain. This representation is not to scale, it's only for illustration purposes.

Conversely, in the frequency domain, signals find description through their power spectral density (PSD), a function outlining how power is distributed across frequency components within the signal. is commonly expressed in watts per hertz (W/Hz), or its logarithmic scale dBW/Hz (1 watt = 0 dBW, 1 milliwatt = ‑30 dBW, 1 microwatt = -60 dBW, 1 nanowatt = -90 dBW, and so on).

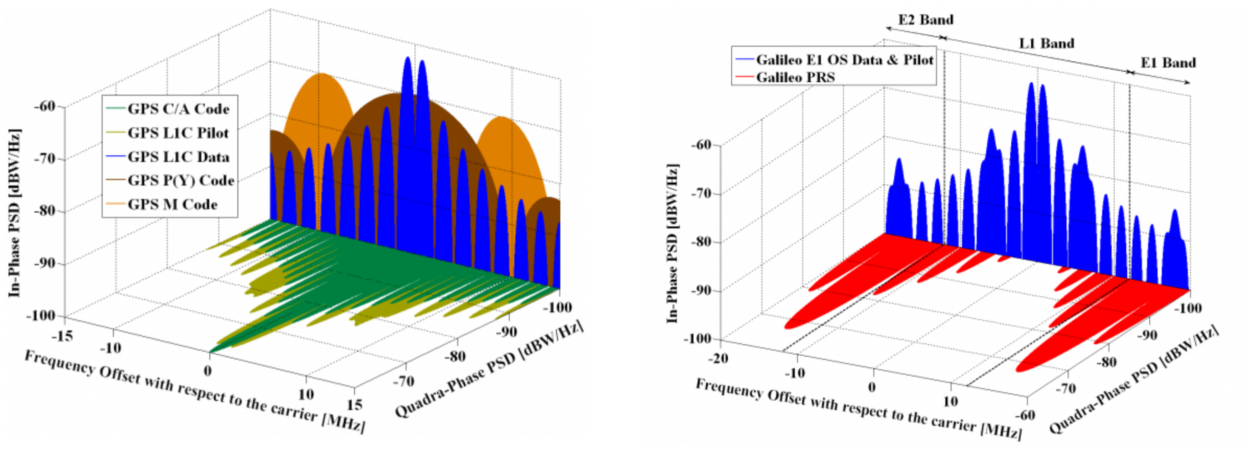


Figure 7 - Signals transmitted by a GNSS satellite expressed in the frequency domain. Various modulations exhibit distinct Power Spectral Density (PSD) functions. Specifically, the figure illustrates the PSD for GPS L1 signals (on the left) and Galileo E1 signals (on the right). Source: Navipedia.

Table 1 presents a comprehensive list of current and planned GNSS signals, each system delivering signals across distinct frequency bands, utilizing diverse modulations, and featuring unique navigation message structures. While most commercial receivers predominantly utilize GPS L1 C/A signals (with some modern devices incorporating Galileo E1b/c), it's noteworthy that only high-end devices, such as those used in surveying, make use of signals from more than one frequency.

GNSS Signal	Center Frequency (MHz)	Modulation type
GPS L5	1176.45	BPSK(10)
Galileo E5a	1176.45	QPSK(10)
BeiDou B2a	1176.45	BPSK(10)
GLONASS L3OC	1202.025	BPSK(10)
Galileo E5b	1207.14	QPSK(10)
BeiDou B2I	1207.14	BPSK(2)
GPS L2C	1227.60	BPSK(1)
GLONASS L2OF	1246.00	BPSK(0.5)
GLONASS L2OC	1248.06	BOC(1,1)
BeiDou B3I	1268.52	BPSK(10)
Galileo E6B	1278.75	BPSK(5)
BeiDou B1I	1561.098	BPSK(2)

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

BeiDou B1C	1575.42	BOC(1,1)
GPS L1 C/A	1575.42	BPSK(1)
GPS L1C	1575.42	BOC(1,1)
Galileo E1b/c	1575.42	CBOC(6,1,1/11)
GLONASS L1OC	1600.995	BOC(1,1)
GLONASS L1OF	1602.00	BPSK(0.5)

Table 1 - Current and planned GNSS signals providing Open Service.

Curious for more in-depth information? Explore the details at https://gnss-sdr.org/docs/tutorials/gnss-signals/. This resource provides a comprehensive description of all available Open Service signals, along with links to the official documentation for further exploration.

3.3 Synchronization of GNSS signals

As already suggested by our musical analogy, it's all about timing: we need to know when the song started. Essentially, this involves determining the locations of the rising edges in the signals illustrated in Figure 6. By doing so, we can sample the signal appropriately, transitioning back to the digital domain, and extract the required information from the received stream of 0s and 1s.

This process of measuring and tracking the evolution of the received signal's timing is termed *synchronization*. It encompasses the estimation of three crucial parameters:

1. **The received frequency**: Although GNSS signals are transmitted by satellites at known, fixed frequencies (for instance, GPS L1 and Galileo E1b/c are transmitted at 1575.42 MHz), those signals are not received at that exact frequency by the receivers on the ground. This discrepancy is due to the Doppler effect−similar to how an ambulance siren sounds at a higher pitch when approaching than when moving away from you. The received frequency (f_{received}) is influenced by the transmitted frequency ($f_{\text{transmitted}}$) and the relative velocity between the satellite and the receiver's antenna:

$$f_{\text{received}} = \frac{c + v_{\text{receiver}}}{c + v_{\text{transmitter}}} f_{\text{transmitted}}$$

where v_{receiver} is the receiver's velocity, $v_{\text{satellite}}$ is the satellite's velocity, and c=299,792,458 m/s is the speed of light.

Financiado por la Unión Europea NextGenerationEU · Plan de Recuperación, Transformación y Resiliencia · UNICO I+D · Gobierno de España

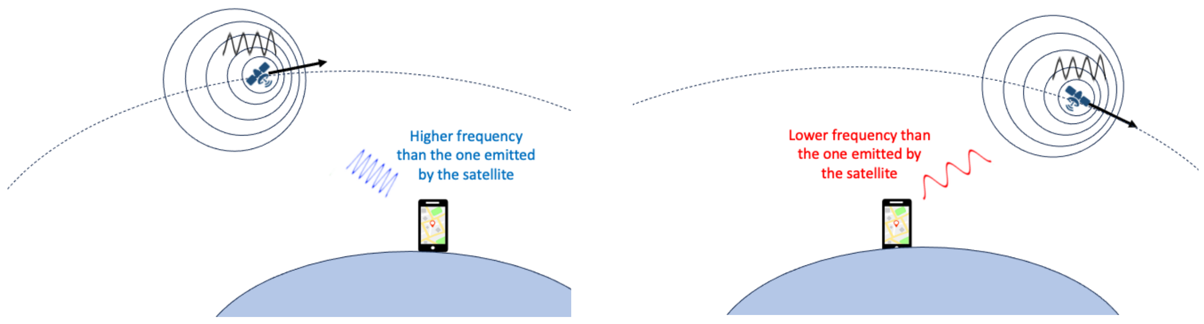

Higher frequency than the one emitted by the satellite
Lower frequency than the one emitted by the satellite

Figure 8 - When a satellite is approaching the receiver, it "sees" a higher frequency than the one originally emitted by the satellite. On the contrary, when the satellite is going away from the receiver, the received frequency is lower. This is called the Doppler effect.

2. **Time delay:** At this point, measuring the propagation delay is not feasible because we don't know how many times the unique code identifying the satellite has been repeated since transmission. However, we can determine when the currently received code started, as we possess knowledge of the full sequence in advance. This time delay is commonly referred to as *code phase*.

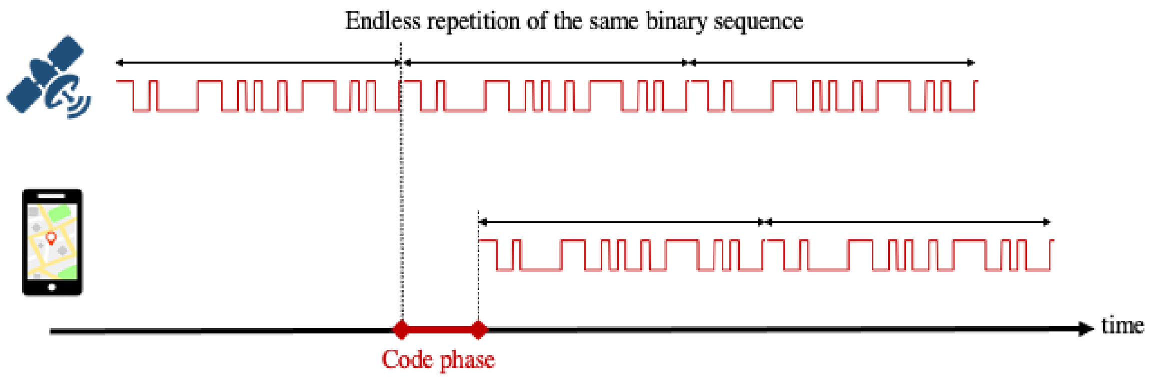


Figure 9 - Concept of code phase.

3. **Carrier phase**: Measurement of the difference between the received signal's carrier frequency and the frequency of the internal oscillator of the receiver.

Determining these three parameters for each of the received signals is the primary task of a GNSS receiver. Once these quantities are accurately estimated and tracked, the receiver can *demodulate* the navigation message, which allows to compute the satellite's position. This step provides all the necessary components to solve the system of equations presented in Figure 4, enabling the determination of the user's receiver position, velocity, and time.

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

3.4 How a GNSS receiver looks like from the inside?

If you disassemble a device equipped with a GNSS receiver, you may identify the chip responsible for its implementation. Typically, it appears as a tiny black box with metallic pads. However, understanding its internal workings or modifying its behaviour is challenging. This scenario changes with a *software-defined* receiver, where the traditional black box is replaced by lines of code. In this setup, you gain visibility into the entire processing chain – from the reception of a stream of raw 0s and 1s to the computation of the user's receiver position, velocity, and time.

The description of the structure and functional blocks of a GNSS receiver provided in this section corresponds to the software architecture implemented in the open-source project GNSS-SDR (see https://gnss-sdr.org). Importantly, it is generic enough to describe the essential design of any GNSS receiver currently available on the market.

GNSS-SDR is a software application that runs in a common personal computer. It provides interfaces through USB and Ethernet buses to a variety of either commercially available or custom-made RF front-ends, adapting the processing algorithms to different sampling frequencies, intermediate frequencies, and sample resolutions. It also can process raw data samples stored in a file. The software performs signal acquisition and tracking of the available satellite signals, decodes the navigation message, and computes the observables needed by positioning algorithms, which ultimately compute the navigation solution.

The functional block diagram is as follows:

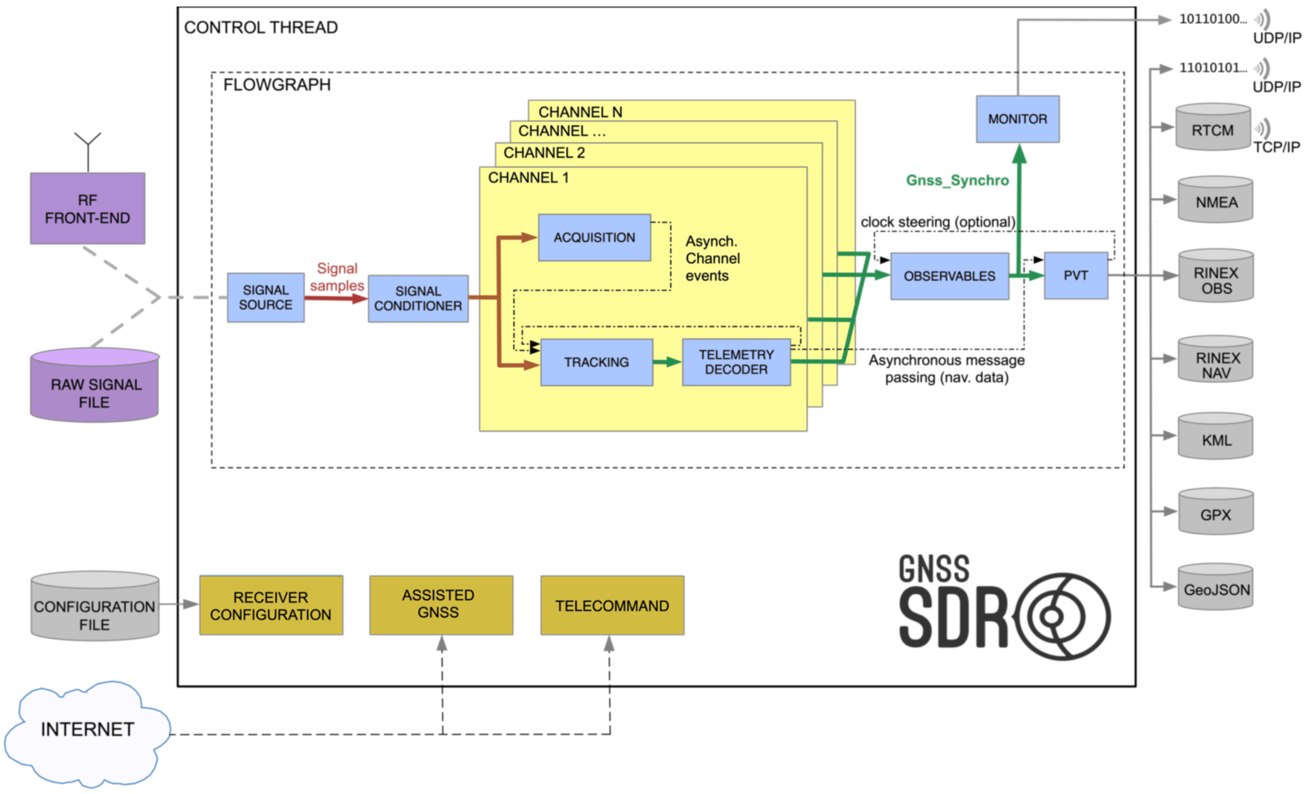


Figure 10 - Basic block diagram of the software-defined GNSS receiver.

The Signal Source is an abstract wrapper that can take either a file containing raw signal samples or be connected to a radiofrequency front-end delivering real-life signal samples at real-time. Then, the stream of samples goes through all the functional blocks (Signal Conditioner, a Channel consisting of an Acquisition, Tracking, and a Telemetry Decoder block, Observables, and PVT), the

Financiado por la Unión Europea NextGenerationEU · Plan de Recuperación, Transformación y Resiliencia · UNICO I+D · GOBIERNO DE ESPAÑA

last one being in charge of computing the position and time of the receiver with the information computed from signals coming from at least four GNSS satellites.

All the Signal Processing Blocks (the blue boxes in Figure 10) can be seen as empty boxes that we need to fill with algorithms and their corresponding parameters. For each of such boxes, we need to specify in the configuration file what will be their specific implementation, how many Channels (that is, how many satellites) our receiver will be able to detect and track at the same time, etc.

In this Workshop, we will learn how to fill a GNSS-SDR configuration file, fully defining our own GNSS receiver, and then running it to obtain Position, Velocity, and Time solutions. Let's first see what the role of each Processing Block is.

Signal Source

A *Signal Source* is the block that injects a continuous stream of raw samples of GNSS signal to the processing flow graph. This is an abstraction that wraps *all* kinds of sources, from samples stored in files (in a variety of formats) to multiple sample streams delivered in real-time by radiofrequency front-ends.

The input of a software receiver are the raw bits that come out from the front-end's analog-to-digital converter (ADC), as sketched in the figure below. Those bits can be read from a file stored in the hard disk or directly in real-time from a hardware device through USB or Ethernet buses.

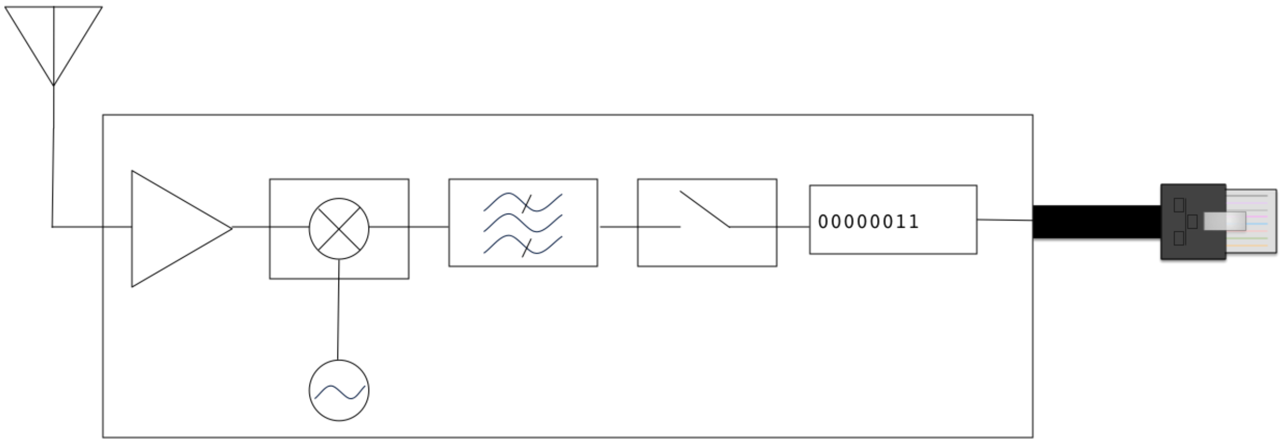


Figure 11 - Simplified block diagram of a generic radio frequency front-end, consisting of an antenna, an amplification stage, downshifting from RF to an intermediate frequency (or baseband), filtering, sampling, and an interface to a host computer for real-time processing mode, or to a storage device for post-processing.

The *Signal Source* block is in charge of implementing the hardware driver, that is, the portion of the code that communicates with the RF front-end and receives the samples coming from the ADC. This communication is usually performed through USB or Ethernet buses. Since real-time processing requires a highly optimized implementation of the whole receiver, this module also allows reading samples from a file stored in a hard disk, and thus processing without time constraints.

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

Signal Conditioner

A *Signal Conditioner* block is in charge of adapting the sample bit depth to a data type tractable at the host computer running the software receiver, and optionally intermediate frequency to baseband conversion, resampling, and filtering.

Regardless of the selected signal source features, the *Signal Conditioner* interface delivers in a unified format a sample data stream to the receiver downstream processing channels, acting as a facade between the signal source and the synchronization channels, providing a simplified interface to the input signal to the downstream blocks.

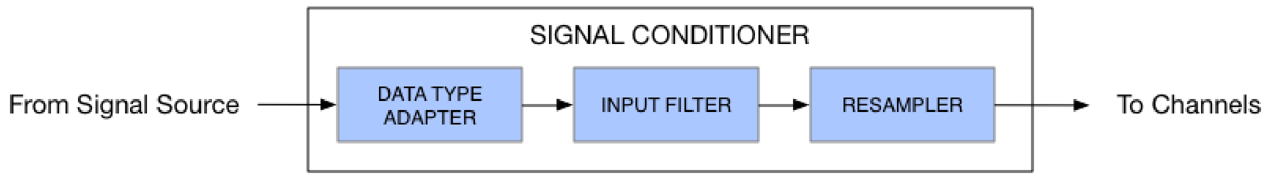


Figure 12 - The Signal Conditioner prepares the input sample stream for further processing.

Channels

Each *Channel* encapsulates blocks for signal acquisition, tracking, and demodulation/decoding of the navigation message for a single satellite. These abstract interfaces can be populated with different algorithms addressing any suitable GNSS signal. The user can define the number of parallel channels *N* to be instantiated by the software receiver.

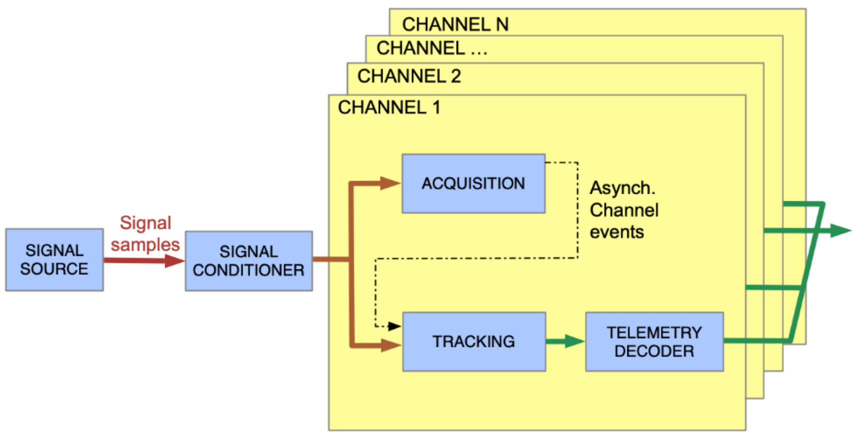


Figure 13 - Each channel can receive one signal band of a single GNSS satellite.

Each channel can receive one signal band of a single GNSS satellite.

Acquisition

The process of identifying the presence of a specific satellite's signal is known as *Signal Acquisition*. Upon positive detection, this stage also necessitates the provision of initial estimates for the time delay and Doppler shift associated with the detected signal.

The search for the existence of a particular satellite signal involves exhaustive testing across all possible combinations of time delay and Doppler shift, resulting in a comprehensive 2D grid search.

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

If the peak value surpasses a predefined detection threshold, the signal is confirmed as present. The coordinates of the peak in the search grid then furnish the initial coarse estimation of the signal's time delay and Doppler shift.

Conversely, in the absence of a peak exceeding the detection threshold, the signal is deemed not present, prompting the receiver to initiate the search for another satellite.

It's important to note that the acquisition process stands out as one of the most computationally demanding stages within the entire processing chain.

Tracking

The role of a *Tracking* block is to follow the evolution of the signal synchronization parameters: code phase, Doppler shift, and carrier phase.

This block receives as an input the stream of samples coming from the Signal Conditioner block. In addition, it has an asynchronous input from the Acquisition block, which informs about the ID of a detected satellite, as well as the rough estimations about the Doppler shift and the delay of the incoming signal.

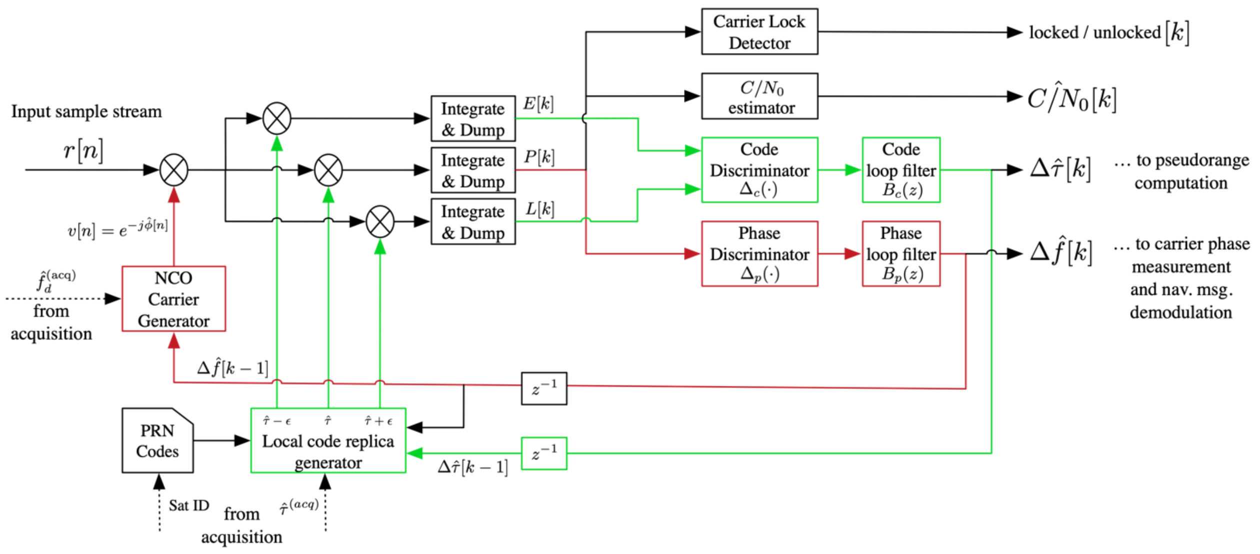


Figure 14 - Architecture of a Delay Lock Loop (DLL), in green, and Phase Lock Loop (PLL), in red, tracking the time evolution of the synchronization parameters.

Telemetry Decoder

The role of a *Telemetry Decoder* block is to obtain the data bits from the navigation data message broadcast by GNSS satellites. More precisely, to demodulate and decode such data from the received signal. Those data bits contain information such as the satellite's position, its clock information, and other system parameters.

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

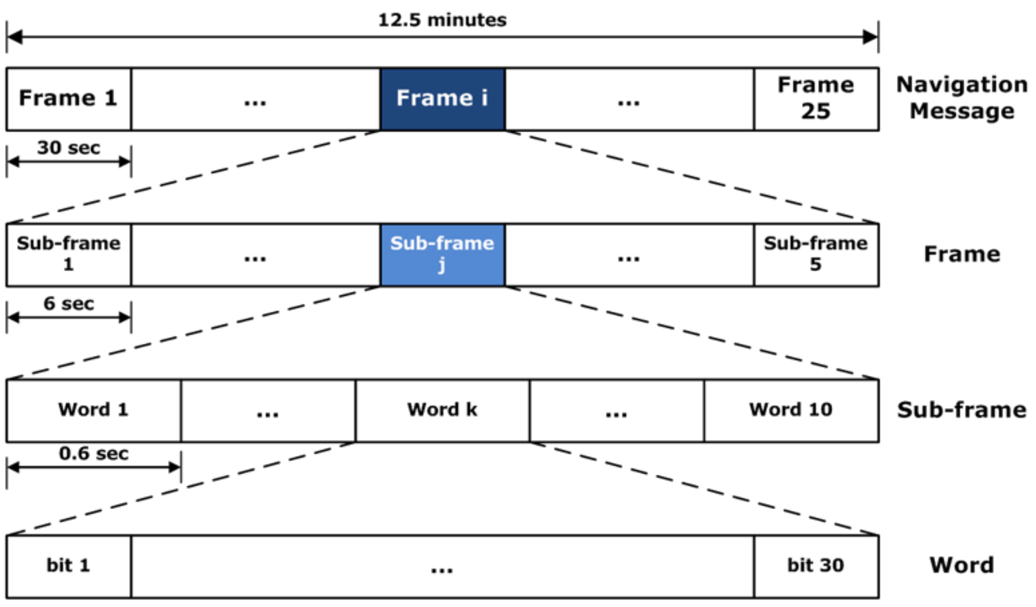


Figure 15 - Structure of the GPS L1 navigation message. Source: Navipedia.

Observables

The role of an *Observables* block is to collect the synchronization data coming from all the processing Channels, and to compute from them the GNSS basic measurements: **pseudorange**, **carrier phase** (or its **phase-range** version), and **Doppler shift** (or its **pseudorange rate** version).

- The **pseudorange measurement** is defined as the difference between the time of reception (expressed in the time frame of the receiver) and the time of transmission (expressed in the time frame of the satellite) of a distinct satellite signal. This corresponds to the distance from the receiver antenna to the satellite antenna, including receiver and satellite clock offsets and other biases, such as atmospheric delays.

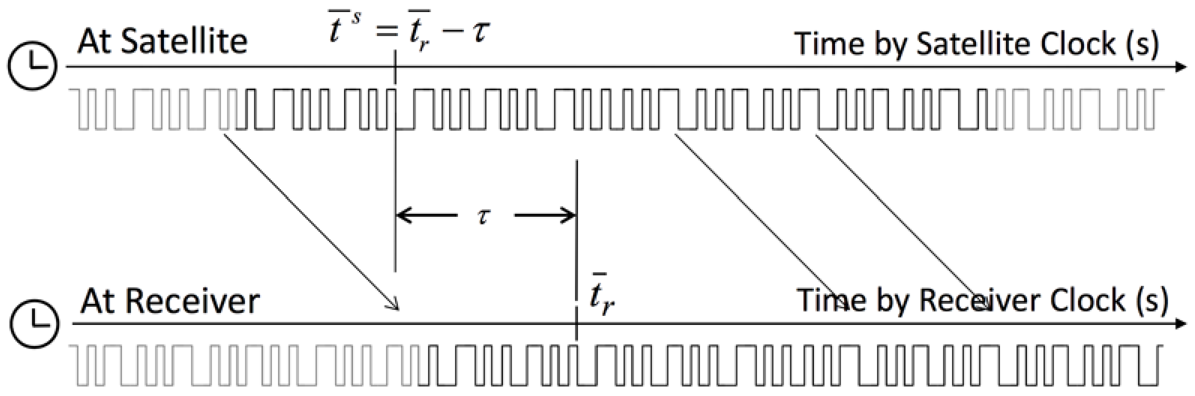

At Satellite
$\bar{t}^s = \bar{t}_r - \tau$
Time by Satellite Clock (s)
τ
At Receiver
$\bar{t}_r$
Time by Receiver Clock (s)

Figure 16 - Concept of pseudorange.

- The **carrier phase measurement** is a measurement of the beat frequency between the received carrier of the satellite signal and a receiver-generated reference frequency.

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

- The **Doppler shift** is the change in frequency for an observer (in this case, the GNSS receiver) moving relative to its source (in this case, a given GNSS satellite).

The set of computed Observables are the input that feeds the algorithm that computes Position, Velocity, and Time in the next processing block.

PVT

The role of a *PVT* block is to compute navigation solutions (that is, receiver's Position, Velocity, and Time) from the Observables, and to deliver the information in adequate formats for further processing or data representation.

By processing signals from multiple satellites, the receiver determines your precise position using mathematical techniques like trilateration or multilateration and taking into account possible sources of error due to various factors like atmospheric disturbances or signal reflections.

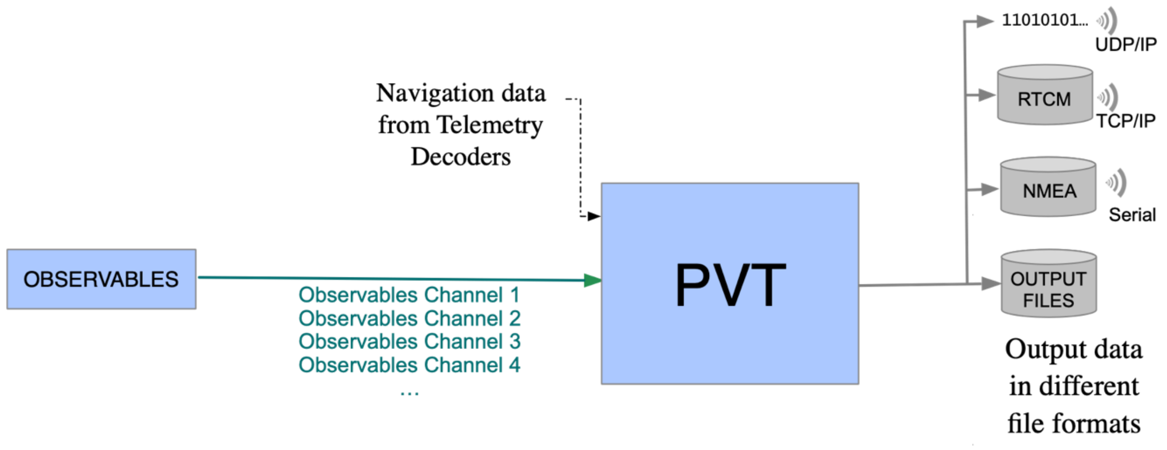


Figure 17 - The PVT block computes the Position, Velocity, and Time solution and prints it in different standard outputs such as KML, GeoJSON, or GPX, among others.

Monitor

The *Monitor* block provides an interface for monitoring the internal status of the receiver in real-time by streaming the receiver's internal data to local or remote clients over UDP. In order to keep things simple, this processing block will not be used in this Workshop.

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

3. Documentation of the GNSS-SDR Signal Processing Blocks

In GNSS-SDR, each configuration file defines a receiver. This page documents the available implementations for each of the *GNSS processing blocks*, represented as blue boxes in the figure below, and their parameters.

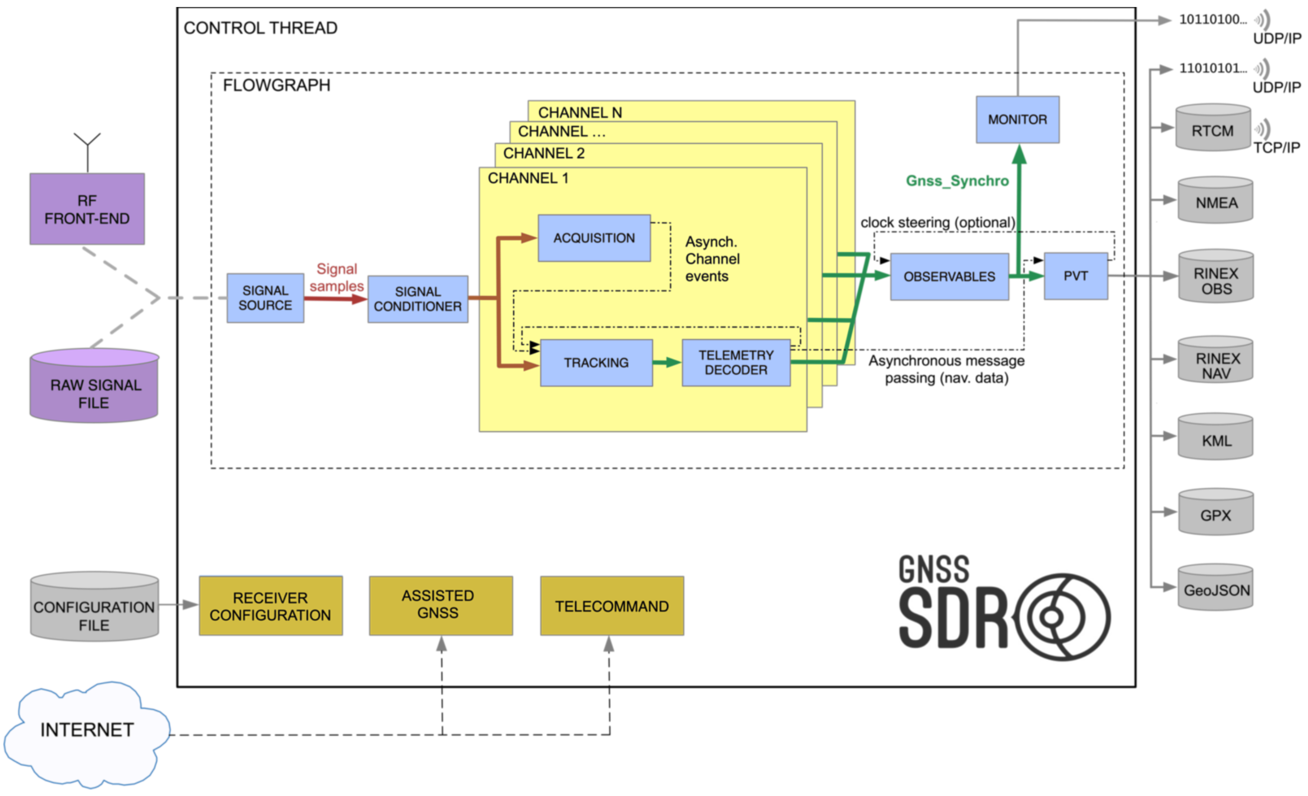


Figure 18. GNSS-SDR signal processing block diagram.

GNSS-SDR configuration files are in the INI format. An INI file is an 8-bit text file in which every property has a name and a value, in the form `name=value`. Properties are case-insensitive, and cannot contain spacing characters. Semicolons (;) indicate the start of a comment; everything between the semicolon and the end of the line is ignored. Example:

```
; THIS IS A COMMENT

SignalConditioner.implementation=Pass_Through ; THIS IS ANOTHER COMMENT
```

In this way, a full GNSS receiver can be uniquely defined in one text file in INI format:

```
$ gnss-sdr --config_file=/path/to/my_receiver.conf
```

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

3.1 Global receiver parameters

The notation is as follows: for a global parameter XXX with value YY, its entry in the configuration file is:

```
GNSS-SDR.XXX=YY
```

If a parameter is not specified in the configuration file, it takes its default value.

Sampling rate of the GNSS baseband engine

In the current design, all the processing Channels need to accept streams of samples at the same rate. This is not necessarily the same as the sampling frequency of the Signal Source, since the Signal Conditioner block can apply some resampling. Please note that this is a **mandatory** parameter, it needs to be present in the configuration file.

Parameter	Description	Required
internal_fs_sps	Input sample rate to the processing channels, in samples per second.	Mandatory
use_acquisition_resampler	[true, false]: If set to true, the Acquisition block makes use of the minimum possible sample rate during the signal acquisition by setting a resampler at its input. This allows reducing the FFT size when using high data rates at internal_fs_sps. All the required setup is configured automatically. This feature is not implemented in all the Acquisition blocks, please check the Acquisition documentation. This parameter defaults to false.	Optional

Global GNSS-SDR parameter: channel's input sampling rate.

Example in the configuration file:

```
GNSS-SDR.internal_fs_sps=4000000
```

Multiple signal sources

Parameter	Description	Required
num_sources	Number of input signal sources. It defaults to 1.	Optional

Example:

```
GNSS-SDR.num_sources=2
```

For more details, please check how to configure multiple signal sources.

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

Banned satellites

By default, GNSS-SDR searches for all the available satellites by PRN identification, from 1 up to the nominal maximum identifier (32 for GPS, 36 for Galileo, and so on). The order in which they are searched for can be altered if assisted GNSS is enabled, but all of them will be eventually processed as long as there are channels available.

The following parameters allow to remove specific satellites from the list of potentially available ones:

Parameter	Description	Required
GPS_banned_prns	List of GPS satellites, by PRN, that will be removed from the list of available satellites and will not be processed. It defaults to empty.	Optional
Galileo_banned_prns	List of Galileo satellites, by PRN, that will be removed from the list of available satellites and will not be processed. It defaults to empty.	Optional
Glonass_banned_prns	List of GLONASS satellites, by PRN, that will be removed from the list of available satellites and will not be processed. It defaults to empty.	Optional
Beidou_banned_prns	List of Beidou satellites, by PRN, that will be removed from the list of available satellites and will not be processed. It defaults to empty.	Optional

With these parameters, users can specify lists of satellites that will not be processed. Satellites on those lists will never be assigned to a processing channel.

Example: since Galileo E14 and E18 satellites are not usable for PVT, they can be removed from the list of Galileo searched satellites by setting:

```
GNSS-SDR.Galileo_banned_prns=14,18
```

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

3.2 Signal Source

A *Signal Source* is the block that injects a continuous stream of raw samples of GNSS signal to the processing flow graph. This is an abstraction that wraps *all* kinds of sources, from samples stored in files (in a variety of formats) to multiple sample streams delivered in real-time by radiofrequency front-ends.

Implementation: File_Signal_Source

This *Signal Source* implementation reads raw signal samples stored in a file, as long as they are stored in one of the following formats: byte, ibyte, short, ishort, float, or gr_complex. Their definition is as follows:

Type name in GNSS-SDR conf files	Definition	Sample stream
byte	Signed integer, 8-bit two's complement number ranging from -128 to 127. C++ type name: int8_t	[S0],[S1],[S2],...
short	Signed integer, 16-bit two's complement number ranging from -32768 to 32767. C++ type name: int16_t	[S0],[S1],[S2],...
float	Defines numbers with fractional parts, can represent values ranging from approx. 1.5✢10✢45 to 3.4✢1038 with a precision of 7 digits (32 bits). C++ type name: float	[S0],[S1],[S2],...
ibyte	Interleaved (I&Q) stream of samples of type byte. C++ type name: int8_t	[S0I],[S0Q],[S1I],[S1Q],[S2I],[S2Q],...
ishort	Interleaved (I&Q) samples of type short. C++ type name: int16_t	[S0I],[S0Q],[S1I],[S1Q],[S2I],[S2Q],...
cbyte	Complex samples, with real and imaginary parts of type byte. C++ type name: lv_8sc_t	[S0I+jS0Q],[S1I+jS1Q],[S2I+jS2Q],...
cshort	Complex samples, with real and imaginary parts of type short. C++ type name: lv_16sc_t	[S0I+jS0Q],[S1I+jS1Q],[S2I+jS2Q],...

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

Type name in GNSS-SDR conf files	Definition	Sample stream
gr_complex	Complex samples, with real and imaginary parts of type float. C++ type name: std::complex<float>	[S0I+jS0Q],[S1I+jS1Q],[S2I+jS2Q],…

Data type definition in GNSS-SDR.

This implementation accepts the following parameters:

Parameter	Description	Required
implementation	File_Signal_Source	Mandatory
filename	Path to the file containing the raw digitized signal samples	Mandatory
sampling_frequency	Sample rate, in samples per second.	Mandatory
samples	Number of samples to be read. If set to 0, the whole file but the last two milliseconds are processed. It defaults to 0.	Optional
item_type	[byte, ibyte, short, ishort, float, gr_complex]: Sample data type. It defaults to gr_complex.	Optional
seconds_to_skip	Seconds of signal to skip from the beginning of the file before start processing. It defaults to 0 s.	Optional
repeat	[true, false]: If set to true, processing of samples restarts the file when the end is reached. It defaults to false.	Optional
enable_throttle_control	[true, false]: If set to true, it places a throttle controlling the data flow. It is generally not required, and it defaults to false.	Optional

Signal Source implementation: File_Signal_Source

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

This implementation assumes that the center frequency is the nominal corresponding to the GNSS frequency band. Any known deviation from that value can be compensated by using the IF parameter of the Freq_Xlating_Fir_Filter implementation of the Input Filter present at the Signal Conditioner block, or later on in the flow graph at the Acquisition and Tracking blocks with their if parameter.

It follows an example of a Signal Source block configured with the File_Signal_Source implementation:

```
;######### SIGNAL_SOURCE CONFIG ############

SignalSource.implementation=File_Signal_Source

SignalSource.filename=/home/user/gnss-sdr/data/my_capture.dat

SignalSource.sampling_frequency=4000000
```

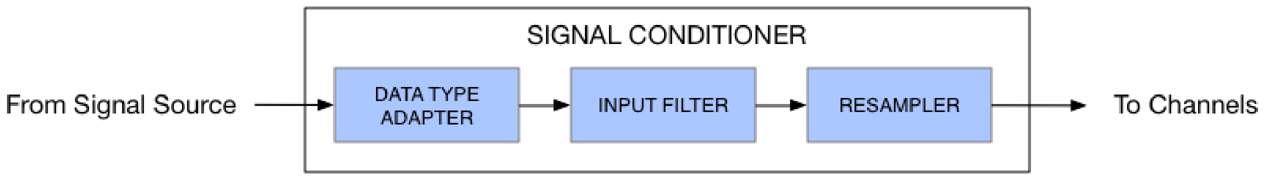
3.3 Signal Conditioner

A *Signal Conditioner* block is in charge of adapting the sample bit depth to a data type tractable at the host computer running the software receiver, and optionally intermediate frequency to baseband conversion, resampling, and filtering.

Regardless of the selected signal source features, the Signal Conditioner interface delivers in a unified format a sample data stream to the receiver downstream processing channels, acting as a facade between the signal source and the synchronization channels, providing a simplified interface to the input signal at a reference, internal sample rate f_{IN}. We denote the complex samples at the Signal Conditioner output as $x_{IN}[n]$. This signal stream feeds a set of parallel Channels.

Implementation: Signal_Conditioner

This implementation is in fact a wrapper for other three processing blocks:



Those inner blocks are in charge of:

- The role of the Data Type Adapter block is to perform a conversion of the data type in the incoming sample stream.
- The role of the Input Filter block is to filter the incoming signal.
- The role of the Resampler block is to resample the signal and to deliver it to the parallel processing channels.
- Any of them can be bypassed by using a Pass_Through implementation.

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

The Signal_Conditioner implementation accepts the following parameters:

Parameter	Description	Required
implementation	Signal_Conditioner	Mandatory
DataTypeAdapter	This implementation requires the configuration of a Data Type Adapter block.	Mandatory
InputFilter	This implementation requires the configuration of an Input Filter block.	Mandatory
Resampler	This implementation requires the configuration of a Resampler block.	Mandatory

Signal Conditioner implementation: Signal_Conditioner.

Implementation: Pass_Through

This implementation copies samples from its input to its output.

It accepts the following parameters:

Parameter	Description	Required
implementation	Pass_Through	Mandatory
item_type	[gr_complex, cshort]: Format of data samples. It defaults to gr_complex.	Optional

Signal Conditioner implementation: Pass_Through.

Examples:

```
;######### SIGNAL_CONDITIONER CONFIG ############

SignalConditioner.implementation=Pass_Through
```

or

```
;######### SIGNAL_CONDITIONER CONFIG ############

SignalConditioner.implementation=Pass_Through
```

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

```
SignalConditioner.item_type=cshort
```

3.4 Data Type Adapter

Documentation for the Data Type Adapter block.

The *Data Type Adapter* is the first processing block inside a *Signal Conditioner* when the latter is using a Signal_Conditioner implementation.

The role of a *Data Type Adapter* block is to perform a conversion of the data type in the sample stream.

This is the first processing block after the Signal Source, and each kind of source can deliver data in different formats.

- If the *Signal Source* is delivering samples at a given intermediate frequency, the *native* data types can be:
    - Real samples: byte, short, float (8, 16, and 32 bits, respectively).
    - Interleaved (I&Q) samples: ibyte, ishort, gr_complex (8+8, 16+16, and 32+32 bits, respectively).
- If the *Signal Source* is delivering samples at baseband, the *native* data types can be:
    - Interleaved (I&Q) samples: ibyte, ishort, gr_complex (8+8, 16+16, and 32+32 bits, respectively).
    - Complex samples: cbyte, cshort, gr_complex (8+8, 16+16, and 32+32 bits, respectively).

This block has several implementations of data type conversions. For more details about sample data types and their usage in GNSS-SDR, please check out our tutorial on data types.

Implementation: Byte_To_Short

This implementation takes samples of type byte (8 bits, real samples) at its input and writes samples of type short (16 bits, real samples) at its output.

It accepts the following parameters:

Parameter	Description	Required
implementation	Byte_To_Short	Mandatory

Data Type Adapter implementation: Byte_To_Short.

Example:

```
;######### DATA_TYPE_ADAPTER CONFIG ############

DataTypeAdapter.implementation=Byte_To_Short
```

Financiado por la Unión Europea NextGenerationEU · Plan de Recuperación, Transformación y Resiliencia · UNICO I+D · GOBIERNO DE ESPAÑA

Implementation: Ibyte_To_Cbyte

This implementation takes samples of type ibyte (interleaved I&Q samples, 8 bits each) at its input and writes samples of type cbyte (complex samples with real and imaginary components of 8 bits each) at its output. This reduces the sample rate by two.

It accepts the following parameters:

Parameter	Description	Required
implementation	Ibyte_To_Cbyte	Mandatory
inverted_spectrum	[true, false]: If set to true, it performs a spectrum inversion. It defaults to false.	Optional

Data Type Adapter implementation: Ibyte_To_Cbyte.

Example:

```
;######### DATA_TYPE_ADAPTER CONFIG ############

DataTypeAdapter.implementation=Ibyte_To_Cbyte
```

Implementation: Ibyte_To_Cshort

This implementation takes samples of type ibyte (interleaved I&Q samples, 8 bits each) at its input and writes samples of type cshort (complex samples with real and imaginary components of 16-bits integers each) at its output. This reduces the sample rate by two.

It accepts the following parameters:

Parameter	Description	Required
implementation	Ibyte_To_Cshort	Mandatory
inverted_spectrum	[true, false]: If set to true, it performs a spectrum inversion. It defaults to false.	Optional

Data Type Adapter implementation: Ibyte_To_Cshort.

Example:

```
;######### DATA_TYPE_ADAPTER CONFIG ############

DataTypeAdapter.implementation=Ibyte_To_Cshort
```

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

Implementation: Ibyte_To_Complex

This implementation takes samples of type ibyte (interleaved I&Q samples, 8 bits each) at its input and writes samples of type gr_complex (complex samples with real and imaginary components of 32 bits each) at its output. This reduces the sample rate by two.

It accepts the following parameters:

Parameter	Description	Required
implementation	Ibyte_To_Complex	Mandatory
inverted_spectrum	[true, false]: If set to true, it performs a spectrum inversion. It defaults to false.	Optional

Data Type Adapter implementation: Ibyte_To_Complex.

Example:

```
;######### DATA_TYPE_ADAPTER CONFIG ############

DataTypeAdapter.implementation=Ibyte_To_Complex
```

Implementation: Ishort_To_Cshort

This implementation takes samples of type ishort (interleaved I&Q samples, 16 bits each) at its input and writes samples of type cshort (complex samples with real and imaginary components of 16 bits each) at its output. This reduces the sample rate by two.

It accepts the following parameters:

Parameter	Description	Required
implementation	Ishort_To_Cshort	Mandatory
inverted_spectrum	[true, false]: If set to true, it performs a spectrum inversion. It defaults to false.	Optional

Data Type Adapter implementation: Ishort_To_Cshort.

Example:

```
;######### DATA_TYPE_ADAPTER CONFIG ############

DataTypeAdapter.implementation=Ishort_To_Cshort
```

Implementation: Ishort_To_Complex

This implementation takes samples of type ishort (interleaved I&Q samples, 16 bits each) at its input and writes samples of type gr_complex (complex samples with real and imaginary components of 32 bits each) at its output. This reduces the sample rate by two.

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

It accepts the following parameters:

Parameter	Description	Required
implementation	Ishort_To_Complex	Mandatory
inverted_spectrum	[true, false]: If set to true, it performs a spectrum inversion. It defaults to false.	Optional

Signal Conditioner implementation: **Ishort_To_Complex**.

Example:

```
;######### DATA_TYPE_ADAPTER CONFIG ############

DataTypeAdapter.implementation=Ishort_To_Complex
```

Implementation: Pass_Through

This implementation copies samples from its input to its output.

It accepts the following parameters:

Parameter	Description	Required
implementation	Pass_Through	Mandatory
item_type	[gr_complex, cshort, cbyte]: Format of data samples. It defaults to gr_complex.	Optional

Data Type Adapter implementation: **Pass_Through**.

Examples:

```
;######### DATA_TYPE_ADAPTER CONFIG ############

DataTypeAdapter.implementation=Pass_Through
```

or

```
;######### DATA_TYPE_ADAPTER CONFIG ############

DataTypeAdapter.implementation=Pass_Through

DataTypeAdapter.item_type=cshort
```

3.5 Input Filter

The *Input Filter* is the second processing block inside a *Signal Conditioner* when the latter is using a Signal_Conditioner implementation.

The role of an *Input Filter* block is to filter noise and possible interferences from the incoming signal.

3.6 Resampler

The *Resampler* is the third processing block inside a *Signal Conditioner* when the latter is using a Signal_Conditioner implementation.

A *Resampler* block is in charge of resampling the signal and delivering it to the N parallel processing channels.

3.7 Channels

Each *Channel* encapsulates blocks for signal acquisition, tracking, and demodulation of the navigation message for a single satellite. These abstract interfaces can be populated with different algorithms addressing any suitable GNSS signal. The user can define the number of parallel channels to be instantiated by the software receiver, and the thread-per-block scheduler imposed by GNU Radio automatically manages the multitasking capabilities of modern multi-core processors. This is done through the configuration file with the Channels_XX.count parameter, where XX is one of the following signal identifiers:

Identifier	Signal	Center Frequency
1G	Glonass L1 C/A	1602.00 MHz
1C	GPS L1 C/A	1575.42 MHz
1B	Galileo E1 B/C	1575.42 MHz
B1	Beidou B1I	1561.098 MHz
E6	Galileo E6B	1278.75 MHz
B3	Beidou B3I	1268.520 MHz
2G	Glonass L2 C/A	1246.00 MHz

Financiado por la Unión Europea NextGenerationEU Plan de Recuperación, Transformación y Resiliencia UNICO I+D GOBIERNO DE ESPAÑA

Identifier	Signal	Center Frequency
2S	GPS L2 L2CM	1227.60 MHz
7X	Galileo E5b	1207.140 MHz
5X	Galileo E5a	1176.450 MHz
L5	GPS L5C	1176.45 MHz

Then, eleven parameters can be set: Channels_1G.count, Channels_1C.count, Channels_1B.count, Channels_B1.count, Channels_E6.count, Channels_B3.count, Channels_2G.count, Channels_2S.count, Channels_5X.count, Channels_7X.count and Channels_L5.count, all of them defaulting to 0.

In addition, the GNSS-SDR flow graph allows setting the number of channels that will be executing signal acquisition (which is known to require a high computational load) concurrently. This is controlled by the parameter Channels.in_acquisition, which defaults to the total number of channels (all of them performing acquisition on different satellite signals at the same time, if required). When working with real-time configurations, it is a good practice to set this parameter to 1 (that is, only one channel performing acquisition at a given time) in order to alleviate the computational burden.

Channels accepts the following parameters:

Parameter	Description	Required
Channels_1G.count	Number of channels targeting Glonass L1 C/A signals. It defaults to 0.	Optional
Channels_1C.count	Number of channels targeting GPS L1 C/A signals. It defaults to 0.	Optional
Channels_1B.count	Number of channels targeting Galileo E1 B/C signals. It defaults to 0.	Optional
Channels_B1.count	Number of channels targeting BeiDou B1I signals. It defaults to 0.	Optional
Channels_E6.count	Number of channels targeting Galileo E6B signals. It defaults to 0.	Optional
Channels_B3.count	Number of channels targeting BeiDou B3I signals. It defaults to 0.	Optional

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

Parameter	Description	Required
Channels_2S.count	Number of channels targeting GPS L2 L2CM signals. It defaults to 0.	Optional
Channels_2G.count	Number of channels targeting Glonass L2 C/A signals. It defaults to 0.	Optional
Channels_7X.count	Number of channels targeting Galileo E5b (I+Q) signals. It defaults to 0.	Optional
Channels_5X.count	Number of channels targeting Galileo E5a (I+Q) signals. It defaults to 0.	Optional
Channels_L5.count	Number of channels targeting GPS L5 signals. It defaults to 0.	Optional
Channel.signal	Assign all channels to a specific signal [1C, 1B, 2S, 5X, L5]. Only required in single-system receivers.	Optional
ChannelN.signal	(where N is the channel number, starting from 0). Assign each channel to a specific signal [1C, 1B, 2S, 5X, L5]. Not required in single-system receivers.	Optional
Channels_1G.RF_channel_ID	Connects channels targeting Glonass L1 C/A to a radio frequency chain. It defaults to 0. Not required in single-band receivers. This feature is present in GNSS-SDR v0.0.18 and later versions.	Optional
Channels_1C.RF_channel_ID	Connects channels targeting GPS L1 C/A to a radio frequency chain. It defaults to 0. Not required in single-band receivers. This feature is present in GNSS-SDR v0.0.18 and later versions.	Optional
Channels_1B.RF_channel_ID	Connects channels targeting Galileo E1 B/C to a radio frequency chain. It defaults to 0. Not required in single-band receivers. This feature is present in GNSS-SDR v0.0.18 and later versions.	Optional
Channels_B1.RF_channel_ID	Connects channels targeting BeiDou B1I to a radio frequency chain. It defaults to 0.Not required in	Optional

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

Parameter	Description	Required
	single-band receivers. This feature is present in GNSS-SDR v0.0.18 and later versions.	
Channels_E6.RF_channel_ID	Connects channels targeting Galileo E6B to a radio frequency chain. It defaults to 0. Not required in single-band receivers. This feature is present in GNSS-SDR v0.0.18 and later versions.	Optional
Channels_B3.RF_channel_ID	Connects channels targeting BeiDou B3I to a radio frequency chain. It defaults to 0. Not required in single-band receivers. This feature is present in GNSS-SDR v0.0.18 and later versions.	Optional
Channels_2S.RF_channel_ID	Connects channels targeting GPS L2 L2CM to a radio frequency chain. It defaults to 0. Not required in single-band receivers. This feature is present in GNSS-SDR v0.0.18 and later versions.	Optional
Channels_2G.RF_channel_ID	Connects channels targeting Glonass L2 C/A to a radio frequency chain. It defaults to 0. Not required in single-band receivers. This feature is present in GNSS-SDR v0.0.18 and later versions.	Optional
Channels_7X.RF_channel_ID	Connects channels targeting Galileo E5b (I+Q) to a radio frequency chain. It defaults to 0. Not required in single-band receivers. This feature is present in GNSS-SDR v0.0.18 and later versions.	Optional
Channels_5X.RF_channel_ID	Connects channels targeting Galileo E5a (I+Q) to a radio frequency chain. It defaults to 0. Not required in single-band receivers. This feature is present in GNSS-SDR v0.0.18 and later versions.	Optional
Channels_L5.RF_channel_ID	Connects channels targeting GPS L5 to a radio frequency chain. It defaults to 0. Not required in single-band receivers. This feature is present in GNSS-SDR v0.0.18 and later versions.	Optional
ChannelN.RF_channel_ID	(where N is the channel number, starting from 0). Connects channel N to a radio frequency chain. Overrides Channels_XX.RF_channel_ID parameter value for a specific channel. It defaults to 0. Not required in single-band receivers.	Optional

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

Parameter	Description	Required
ChannelN.Signal_Source_ID	(where N is the channel number, starting from 0). Connects channel N to a signal source. It defaults to 0. Not required in single-source receivers.	Optional
ChannelN.satellite	(where N is the channel number, starting from 0). Assigns channel N to given satellite by its PRN. This channel will always be trying to acquire and track the given satellite.	Optional
Channels.in_acquisition	Maximum number of channels performing signal acquisition at the same time. The recommended value is 1. In the case of having assigned a channel to a given satellite (e.g., with Channel0.satellite=1), it is recommended to increase this number in order to always have at least one channel searching for new satellites. It defaults to the total number of channels.	Optional

Then, each type of defined channel requires the configuration of:

- *Acquisition* blocks targeting the desired signal type, in charge of the detection of signals coming from a given GNSS satellite and, in the case of a positive detection, to provide coarse estimations of the code phase and the Doppler shift,
- *Tracking* blocks targeting the desired signal type, in charge of following the evolution of the signal synchronization parameters: code phase, Doppler shift, and carrier phase; and
- *Telemetry Decoder* blocks targeting the desired signal type, in charge of demodulating and decoding the GNSS navigation message carried by that particular signal.

Examples for different receiver architectures are provided below.

Single system, single band receiver

Setting a single-band receiver with twelve channels devoted to GPS L1 C/A signal can be done as:

```
;######### CHANNELS GLOBAL CONFIG ############

Channels_1C.count=12

Channel.signal=1C

Channels.in_acquisition=1


Acquisition_1C.implementation=...

; or Acquisition_1C0, ..., Acquisition_1C11, and parameters.
```

Financiado por la Unión Europea NextGenerationEU Plan de Recuperación, Transformación y Resiliencia UNICO I+D GOBIERNO DE ESPAÑA

```
Tracking_1C.implementation=...
; or Tracking_1C0, ..., Tracking_1C11, and parameters.

TelemetryDecoder_1C.implementation=...
; or TelemetryDecoder_1C0, ..., TelemetryDecoder_1C11, and parameters.
```

Multi-constellation, single band receiver

When defining a multi-system receiver, the user must specify which channels are devoted to each signal. This is done through the parameter ChannelN.signal, where N is the absolute channel number, starting from zero:

```
;######### CHANNELS CONFIG ############
Channels_1C.count=4
Channels_1B.count=4
Channels.in_acquisition=1
Channel0.signal=1C
Channel1.signal=1C
Channel2.signal=1C
Channel3.signal=1C
Channel4.signal=1B
Channel5.signal=1B
Channel6.signal=1B
Channel7.signal=1B

Acquisition_1C.implementation=...
; or Acquisition_1C0, ..., Acquisition_1C3, and parameters.
Acquisition_1B.implementation=...
; or Acquisition_1B4, ..., Acquisition_1B7, and parameters.

Tracking_1C.implementation=...
; or Tracking_1C0, ..., Tracking_1C3
Tracking_1B.implementation=...
; or Tracking_1B4, ..., Tracking_1B7
```

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

```
TelemetryDecoder_1C.implementation=...
; or TelemetryDecoder_1C0, ..., TelemetryDecoder_1C3, and parameters.
TelemetryDecoder_1B.implementation=...
; or TelemetryDecoder_1B4, ..., TelemetryDecoder_1B7, and parameters.
```

3.8 Acquisition

The role of an *Acquisition* block is the detection of the presence/absence of signals coming from a given GNSS satellite. In the case of a positive detection, it should provide coarse estimations of the code phase and the Doppler shift, yet accurate enough to initialize the delay and phase tracking loops.

This implementation accepts the following parameters:

Global Parameter	Description	Required
GNSS-SDR.internal_fs_sps	Input sample rate to the processing channels, in samples per second.	Mandatory
GNSS-SDR.use_acquisition_resampler	[true, false]: If set to true, the Acquisition block makes use of the minimum possible sample rate during acquisition by setting a resampler at its input. This allows reducing the FFT size when using high data rates at GNSS-SDR.internal_fs_sps. All the required setup is configured automatically. It defaults to false.	Optional

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

Parameter	Description	Required
implementation	GPS_L1_CA_PCPS_Acquisition	Mandatory
item_type	[gr_complex, cshort, cbyte]: Set the sample data type expected at the block input. It defaults to gr_complex.	Optional
doppler_max	Maximum Doppler value in the search grid, in Hz. It defaults to 5000 Hz.	Optional
doppler_step	Frequency step in the search grid, in Hz. It defaults to 500 Hz.	Optional
threshold	Decision threshold from which a signal will be considered present. It defaults to 0.0 (*i.e.*, all signals are declared present).	Optional
pfa	If defined, it supersedes the threshold value and computes a new threshold based on the Probability of False Alarm. It defaults to 0.0 (*i.e.*, not set).	Optional
coherent_integration_time_ms	Set the integration time , in ms. It defaults to 1 ms.	Optional
bit_transition_flag	[true, false]: If set to true, it takes into account the possible presence of a bit transition, so the effective integration time is doubled. When set, it invalidates the value of max_dwells. It defaults to false.	Optional
max_dwells	Set the maximum number of non-coherent dwells to declare a signal present. It defaults to 1.	Optional
repeat_satellite	[true, false]: If set to true, the block will search again for the same satellite once its presence has been discarded. Useful for testing. It defaults to false.	Optional
blocking	[true, false]: If set to false, the acquisition workload is executed in a separate thread, outside the GNU Radio scheduler that manages the flow graph, and the block skips over samples that arrive while the processing thread is busy. This is especially useful in real-time operation using radio frequency front-ends,	Optional

Financiado por la Unión Europea NextGenerationEU · Plan de Recuperación, Transformación y Resiliencia · UNICO I+D · GOBIERNO DE ESPAÑA

Parameter	Description	Required
	overcoming the processing bottleneck for medium and high sampling rates. However, this breaks the determinism provided by the GNU Radio scheduler, and different processing results can be obtained in different machines. Do not use this option for file processing. It defaults to true.	
make_two_steps	[true, false]: If set to true, an acquisition refinement stage is performed after a signal is declared present. This allows providing an updated, refined Doppler estimation to the Tracking block. It defaults to false.	Optional
second_nbins	If make_two_steps is set to true, this parameter sets the number of bins done in the acquisition refinement stage. It defaults to 4.	Optional
second_doppler_step	If make_two_steps is set to true, this parameter sets the Doppler step applied in the acquisition refinement stage, in Hz. It defaults to 125 Hz.	Optional
dump	[true, false]: If set to true, it enables the Acquisition internal binary data file logging. It defaults to false.	Optional
dump_filename	If dump is set to true, name of the file in which internal data will be stored. This parameter accepts either a relative or an absolute path; if there are non-existing specified folders, they will be created. It defaults to ./acquisition, so files with name ./acquisition_G_1C_ch_N_K_sat_P.mat (where N is the channel number defined by dump_channel, K is the dump number, and P is the targeted satellite's PRN number) will be generated.	Optional
dump_channel	If dump is set to true, channel number from which internal data will be stored. It defaults to 0.	Optional

Acquisition implementation: GPS_L1_CA_PCPS_Acquisition.

Example:

```
;######### ACQUISITION CONFIG FOR GPS L1 CHANNELS ############

Acquisition_1C.implementation=GPS_L1_CA_PCPS_Acquisition

Acquisition_1C.doppler_max=5000
```

Financiado por la Unión Europea NextGenerationEU · Plan de Recuperación, Transformación y Resiliencia · UNICO I+D · GOBIERNO DE ESPAÑA

```
Acquisition_1C.doppler_step=250

Acquisition_1C.pfa=0.01

Acquisition_1C.coherent_integration_time_ms=1

Acquisition_1C.max_dwells=1
```

3.9 Tracking

The role of a *Tracking* block is to follow the evolution of the signal synchronization parameters: code phase, Doppler shift, and carrier phase.

This implementation accepts the following parameters:

Global Parameter	Description	Required
GNSS-SDR.internal_fs_sps	Input sample rate to the processing channels, in samples per second.	Mandatory

Parameter	Description	Required
implementation	GPS_L1_CA_DLL_PLL_Tracking	Mandatory
item_type	[gr_complex]: Set the sample data type expected at the block input. It defaults to gr_complex.	Optional
extend_correlation_symbols	Sets the number of correlation symbols to be extended after bit synchronization has been achieved. Each symbol is 1 ms, so setting this parameter to 20 means a coherent integration time of 20 ms. Each bit is 20 ms, so the value of this parameter must be a divisor of it (*e.g.*, 2, 4, 5, 10, 20). The higher this parameter is, the better local clock stability will be required. It defaults to 1.	Optional
pll_bw_hz	Bandwidth of the PLL low-pass filter, in Hz. It defaults to 50 Hz.	Optional
pll_bw_narrow_hz	Bandwidth of the PLL low-pass filter after bit synchronization, in Hz. It defaults to 20 Hz.	Optional

Financiado por la Unión Europea NextGenerationEU Plan de Recuperación, Transformación y Resiliencia UNICO I+D GOBIERNO DE ESPAÑA

Parameter	Description	Required
pll_filter_order	[2, 3]. Sets the order of the PLL low-pass filter. It defaults to 3.	Optional
enable_fll_pull_in	[true, false]. If set to true, enables the FLL during the pull-in time. It defaults to false.	Optional
enable_fll_steady_state	[true, false]. If set to true, the FLL is enabled beyond the pull-in stage. It defaults to false.	Optional
fll_bw_hz	Bandwidth of the FLL low-pass filter, in Hz. It defaults to 35 Hz.	Optional
pull_in_time_s	Time, in seconds, in which the tracking loop will be in pull-in mode. It defaults to 2 s.	Optional
dll_bw_hz	Bandwidth of the DLL low-pass filter, in Hz. It defaults to 2 Hz.	Optional
dll_bw_narrow_hz	Bandwidth of the DLL low-pass filter after bit synchronization, in Hz. It defaults to 2 Hz.	Optional
dll_filter_order	[1, 2, 3]. Sets the order of the DLL low-pass filter. It defaults to 2.	Optional
early_late_space_chips	Spacing between Early and Prompt and between Prompt and Late correlators, normalized by the chip period . It defaults to 0.5.	Optional
early_late_space_narrow_chips	Spacing between Early and Prompt and between Prompt and Late correlators, normalized by the chip period , after bit synchronization. It defaults to 0.5.	Optional
carrier_aiding	[true, false]. If set to true, the code loop is aided by the carrier loop. It defaults to true.	Optional

Financiado por la Unión Europea NextGenerationEU Plan de Recuperación, Transformación y Resiliencia UNICO I+D GOBIERNO DE ESPAÑA

Parameter	Description	Required
cn0_samples	Number of correlator outputs used for CN0 estimation. It defaults to 20.	Optional
cn0_min	Minimum valid CN0 (in dB-Hz). It defaults to 25 dB-Hz.	Optional
max_lock_fail	Maximum number of lock failures before dropping a satellite. It defaults to 50.	Optional
carrier_lock_th	Carrier lock threshold (in rad). It defaults to 0.85 rad.	Optional
cn0_smoother_samples	Number of samples used to smooth the value of the estimated / 0. It defaults to 200 samples.	Optional
cn0_smoother_alpha	Forgetting factor of the / 0 smoother, as in = +(1✥) ✥1. It defaults to 0.002.	Optional
carrier_lock_test_smoother_samples	Number of samples used to smooth the value of the carrier lock test. It defaults to 25 samples.	Optional
carrier_lock_test_smoother_alpha	Forgetting factor of the carrier lock detector smoother, as in = +(1✥) ✥1. It defaults to 0.002.	Optional
dump	[true, false]: If set to true, it enables the Tracking internal binary data file logging, in form of ".dat" files. This format can be retrieved and plotted in Matlab / Octave, see scripts under gnss-sdr/src/utils/matlab/. It defaults to false.	Optional
dump_filename	If dump is set to true, name of the file in which internal data will be stored. This parameter accepts either a relative or an absolute path; if there are non-existing specified folders, they will be created. It defaults to ./track_ch, so files in the form "./track_chX.dat", where X is the channel number, will be generated.	Optional

Financiado por la Unión Europea NextGenerationEU · Plan de Recuperación, Transformación y Resiliencia · UNICO I+D · GOBIERNO DE ESPAÑA

Parameter	Description	Required
dump_mat	[true, false]. If dump=true, when the receiver exits it can convert the ".dat" files stored by this block into ".mat" files directly readable from Matlab and Octave. If the receiver has processed more than a few minutes of signal, this conversion can take a long time. In systems with limited resources, you can turn off this conversion by setting this parameter to false. It defaults to true, so ".mat" files are generated by default if dump=true.	Optional

Tracking implementation: GPS_L1_CA_DLL_PLL_Tracking.

Example:

```
;######### TRACKING CONFIG FOR GPS L1 CHANNELS ############

Tracking_1C.implementation=GPS_L1_CA_DLL_PLL_Tracking

Tracking_1C.item_type=gr_complex

Tracking_1C.extend_correlation_symbols=20

Tracking_1C.early_late_space_chips=0.5;

Tracking_1C.early_late_space_narrow_chips=0.1;

Tracking_1C.pll_bw_hz=35;

Tracking_1C.dll_bw_hz=2.0;

Tracking_1C.pll_bw_narrow_hz=5.0;

Tracking_1C.dll_bw_narrow_hz=0.50;

Tracking_1C.fll_bw_hz=10

Tracking_1C.enable_fll_pull_in=true;

Tracking_1C.enable_fll_steady_state=false

Tracking_1C.dump=false

Tracking_1C.dump_filename=tracking_ch_
```

Financiado por la Unión Europea NextGenerationEU Plan de Recuperación, Transformación y Resiliencia UNICO I+D GOBIERNO DE ESPAÑA

3.10 Telemetry Decoder

The role of a *Telemetry Decoder* block is to obtain the data bits from the navigation message broadcast by GNSS satellites.

This implementation accepts the following parameters:

Parameter	Description	Required
implementation	GPS_L1_CA_Telemetry_Decoder	Mandatory
dump	[true, false]: If set to true, it enables the Telemetry Decoder internal binary data file logging It defaults to false.	Optional
dump_filename	If dump is set to true, base name of the files in which internal data will be stored. It defaults to ./telemetry, so files will be named ./telemetryN, where N is the channel number (automatically added).	Optional
remove_mat	[true, false]: If dump is set to true, the binary output is converted to .mat format, readable from Matlab7octave and Python, at the end of the receiver execution. By default, it is set to the same value as dump.	Optional
remove_dat	[true, false]: If dump=true and dump_mat is not set, or set to true, then this parameter controls if the internal .dat binary file is removed after conversion to .mat, leaving a cleaner output if the user is not interested in the .dat file. By default, this parameter is set to false.	Optional
dump_crc_stats	[true, false]: If set to true, the success rate of the CRC check when decoding navigation messages is reported in a file generated at the end of the processing (or when exiting with q + [Enter]). By default, this parameter is set to false.	Optional
dump_crc_stats_filename	If dump_crc_stats=true, this parameter sets the base name of the files in which the CRC success rate is reported. It defaults to telemetry_crc_stats, so files named telemetry_crc_stats_chN.txt will be created, with N in chN being the channel number.	Optional

Financiado por la Unión Europea NextGenerationEU Plan de Recuperación, Transformación y Resiliencia UNICO I+D GOBIERNO DE ESPAÑA

Example

```
;######### TELEMETRY DECODER CONFIG FOR GPS L1 CHANNELS ############
TelemetryDecoder_1C.implementation=GPS_L1_CA_Telemetry_Decoder
TelemetryDecoder_1C.dump=false
```

3.11 Observables

The role of an *Observables* block is to collect the synchronization data coming from all the processing Channels, and to compute from them the GNSS basic measurements: **pseudorange**, **carrier phase** (or its **phase-range** version), and **Doppler shift** (or its **pseudorange rate** version).

This implementation computes observables by collecting the outputs of channels for all kinds of allowed GNSS signals. **You always can use this implementation in your configuration file, since it accepts all kind of (single- or multi-band, single- or multi-constellation) receiver configurations.**

It accepts the following parameters:

Parameter	Description	Required
implementation	Hybrid_Observables	Mandatory
enable_carrier_smoothing	[true, false]: If set to true, it enables carrier smoothing of code pseudoranges. It defaults to false.	Optional
smoothing_factor	If enable_carrier_smoothing is set to true, this parameter sets the smoothing factor \(M\) (see equation (\(\ref{eq:smoothing}\))). It defaults to 200.	Optional
dump	[true, false]: If set to true, it enables the Observables internal binary data file logging. Storage in .mat files readable from Matlab, Octave and Python is available starting from GNSS-SDR v0.0.10, see below. It defaults to false.	Optional
dump_filename	If dump is set to true, name of the file in which internal data will be stored. This parameter accepts	Optional

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

Parameter	Description	Required
	either a relative or an absolute path; if there are non-existing specified folders, they will be created. It defaults to ./observables.dat	
dump_mat	[true, false]. If dump=true, when the receiver exits it can convert the ".dat" files stored by this block into ".mat" files directly readable from Matlab and Octave. If the receiver has processed more than a few minutes of signal, this conversion can take a long time. In systems with limited resources, you can turn off this conversion by setting this parameter to false. It defaults to true, so ".mat" files are generated by default if dump=true.	Optional

Observables implementation: Hybrid_Observables.

Example:

```
;######### OBSERVABLES CONFIG ############

Observables.implementation=Hybrid_Observables

Observables.dump=false
```

3.12 PVT

The role of a *PVT* block is to compute navigation solutions and deliver information in adequate formats for further processing or data representation.

This implementation accepts the following parameters:

Global Parameter	**Description**	**Required**
GNSS-SDR.SUPL_gps_ephemeris_xml	Name of an XML file containing GPS ephemeris data. It defaults to ./gps_ephemeris.xml	Optional
GNSS-SDR.pre_2009_file	[true, false]: If you are processing raw data files containing GPS L1 C/A signals dated before July 14, 2009, you can set this parameter to true in	Optional

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

Global Parameter	Description	Required
	order to get the right date and time. It defaults to false.	

Parameter	Description	Required
implementation	RTKLIB_PVT	Mandatory
output_rate_ms	Rate at which PVT solutions will be computed, in ms. The minimum is 20 ms, and the value must be a multiple of it. It defaults to 500 ms.	Optional
display_rate_ms	Rate at which PVT solutions will be displayed in the terminal, in ms. It must be multiple of output_rate_ms. It defaults to 500 ms.	Optional
positioning_mode	[Single, PPP_Static, PPP_Kinematic] Set positioning mode. Single: Single point positioning. PPP_Static: Precise Point Positioning with static mode. PPP_Kinematic: Precise Point Positioning for a moving receiver. It defaults to Single.	Optional
num_bands	[1: L1 Single frequency, 2: L1 and L2 Dual-frequency, 3: L1, L2 and L5 Triple-frequency] This option is automatically configured according to the Channels configuration. This option can be useful to force some configuration (*e.g.*, single-band solution in a dual-frequency receiver).	Optional
elevation_mask	Set the elevation mask angle, in degrees. It defaults to 15 .	Optional
dynamics_model	[0: Off, 1: On] Set the dynamics model of the receiver. If set to 1 and PVT.positioning_mode=PPP_Kinematic, the receiver position is predicted with the estimated velocity and acceleration. It defaults to 0 (no dynamics model).	Optional

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

Parameter	Description	Required
iono_model	[OFF, Broadcast, Iono-Free-LC]. Set ionospheric correction options. OFF: Not apply the ionospheric correction. Broadcast: Apply broadcast ionospheric model. Iono-Free-LC: Ionosphere-free linear combination with dual-frequency (L1-L2 for GPS or L1-L5 for Galileo) measurements is used for ionospheric correction. It defaults to OFF (no ionospheric correction)	Optional
trop_model	[OFF, Saastamoinen, Estimate_ZTD, Estimate _ZTD_Grad]. Set whether tropospheric parameters (zenith total delay at rover and base-station positions) are estimated or not. OFF: Not apply troposphere correction. Saastamoinen: Apply Saastamoinen model. Estimate_ZTD: Estimate ZTD (zenith total delay) parameters as EKF states. Estimate_ZTD_Grad: Estimate ZTD and horizontal gradient parameters as EKF states. If defaults to OFF (no troposphere correction).	Optional
enable_rx_clock_correction	[true, false]: If set to true, the receiver makes use of the PVT solution to correct timing in observables, hence providing continuous measurements in long observation periods. If set to false, the Time solution is only used in the computation of Observables when the clock offset estimation exceeds the value of max_clock_offset_ms. This parameter defaults to false.	Optional
max_clock_offset_ms	If enable_rx_clock_correction is set to false, this parameter sets the maximum allowed local clock offset with respect to the Time solution. If the estimated offset exceeds this parameter, a clock correction is applied to the computation of Observables. It defaults to 40 ms.	Optional

Financiado por la Unión Europea NextGenerationEU Plan de Recuperación, Transformación y Resiliencia UNICO I+D GOBIERNO DE ESPAÑA

Parameter	Description	Required
code_phase_error_ratio_l1	Code/phase error ratio for the L1 band. It defaults to 100.	Optional
carrier_phase_error_factor_a	Carrier phase error factor 2. It defaults to 0.003 m.	Optional
carrier_phase_error_factor_b	Carrier phase error factor 2. It defaults to 0.003 m.	Optional
slip_threshold	Set the cycle-slip threshold (m) of geometry-free LC carrier-phase difference between epochs. It defaults to 0.05.	Optional
threshold_reject_GDOP	Set the reject threshold of GDOP. If the GDOP is over the value, the observable is excluded for the estimation process as an outlier. It defaults to 30.0.	Optional
threshold_reject_innovation	Set the reject threshold of innovation (pre-fit residual) (m). If the innovation is over the value, the observable is excluded for the estimation process as an outlier. It defaults to 30.0 m.	Optional
number_filter_iter	Set the number of iteration in the measurement update of the estimation filter. If the baseline length is very short like 1 m, the iteration may be effective to handle the nonlinearity of the measurement equation. It defaults to 1.	Optional
sigma_bias	Set the process noise standard deviation of carrier-phase bias , in cycles/ . It defaults to 0.0001 cycles/ .	Optional
sigma_trop	Set the process noise standard deviation of zenith tropospheric delay , in m/ . It defaults to 0.0001 m/ .	Optional

Financiado por la Unión Europea NextGenerationEU · Plan de Recuperación, Transformación y Resiliencia · UNICO I+D · GOBIERNO DE ESPAÑA

Parameter	Description	Required
raim_fde	[0, 1]: Set whether RAIM (receiver autonomous integrity monitoring) FDE (fault detection and exclusion) feature is enabled or not. It defaults to 0 (RAIM not enabled)	Optional
reject_GPS_IIA	[0, 1]: Set whether the GPS Block IIA satellites are excluded or not. Those satellites often degrade the PPP solutions due to unpredicted behavior of yaw-attitude. It defaults to 0 (no rejection).	Optional
phwindup	[0, 1]: Set whether the phase windup correction for PPP modes is applied or not. It defaults to 0 (no phase windup correction).	Optional
earth_tide	[0, 1]: Set whether earth tides correction is applied or not. If set to 1, the solid earth tides correction , is applied to the PPP solution, following the description in IERS Technical Note No. 32[3], Chapter 7. It defaults to 0 (no Earth tide correction).	Optional
output_enabled	[true, false]: If set to false, output data files are not stored. It defaults to true.	Optional
rtcm_output_file_enabled	[true, false]: If set to false, RTCM binary files are not stored. It defaults to false.	Optional
gpx_output_enabled	[true, false]: If set to false, GPX files are not stored. It defaults to output_enabled.	Optional
geojson_output_enabled	[true, false]: If set to false, GeoJSON files are not stored. It defaults to output_enabled.	Optional
kml_output_enabled	[true, false]: If set to false, KML files are not stored. It defaults to output_enabled.	Optional

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

Parameter	**Description**	**Required**
xml_output_enabled	[true, false]: If set to false, XML files are not stored. It defaults to output_enabled.	Optional
rinex_output_enabled	[true, false]: If set to false, RINEX files are not stored. It defaults to output_enabled.	Optional
rinex_version	[2: version 2.11, 3: version 3.02] Version of the generated RINEX files. It defaults to 3.	Optional
rinex_name	Sets the base name of the RINEX files. If this parameter is not specified, a default one will be assigned. The command-line flag --RINEX_name, if present, overrides this parameter.	Optional
rinexobs_rate_ms	Rate at which observations are annotated in the RINEX file, in ms. The minimum is 20 ms, and must be a multiple of output_rate_ms. It defaults to 1000 ms.	Optional
nmea_output_file_enabled	[true, false]: If set to false, NMEA sentences are not stored. It defaults to true.	Optional
nmea_dump_filename	Name of the file containing the generated NMEA sentences in ASCII format. It defaults to ./nmea_pvt.nmea.	Optional
flag_nmea_tty_port	[true, false]: If set to true, the NMEA sentences are also sent to a serial port device. It defaults to false.	Optional
nmea_dump_devname	If flag_nmea_tty_port is set to true, descriptor of the serial port device. It defaults to /dev/tty1.	Optional
flag_rtcm_server	[true, false]: If set to true, it runs up a TCP server that is serving RTCM messages to the	Optional

Financiado por la Unión Europea NextGenerationEU · Plan de Recuperación, Transformación y Resiliencia · UNICO I+D · GOBIERNO DE ESPAÑA

Parameter	Description	Required
	connected clients during the execution of the software receiver. It defaults to false.	
rtcm_tcp_port	If flag_rtcm_server is set to true, TCP port from which the RTCM messages will be served. It defaults to 2101.	Optional
rtcm_station_id	Station ID reported in the generated RTCM messages. It defaults to 1234.	Optional
rtcm_MT1045_rate_ms	Rate at which RTCM Message Type 1045 (Galileo Ephemeris data) will be generated, in ms. If set to 0, mutes this message. It defaults to 5000 ms.	Optional
rtcm_MT1019_rate_ms	Rate at which RTCM Message Type 1019 (GPS Ephemeris data) will be generated, in ms. If set to 0, mutes this message. It defaults to 5000 ms.	Optional
rtcm_MSM_rate_ms	Default rate at which RTCM Multiple Signal Messages will be generated. It defaults to 1000 ms.	Optional
rtcm_MT1077_rate_ms	Rate at which RTCM Multiple Signal Messages GPS MSM7 (MT1077 - Full GPS observations) will be generated, in ms. If set to 0, mutes this message. It defaults to rtcm_MSM_rate_ms.	Optional
rtcm_MT1097_rate_ms	Rate at which RTCM Multiple Signal Messages Galileo MSM7 (MT1097 - Full Galileo observations) will be generated, in ms. If set to 0, mutes this message. It defaults to rtcm_MSM_rate_ms.	Optional
flag_rtcm_tty_port	[true, false]: If set to true, the generated RTCM messages are also sent to a serial port device. It defaults to false.	Optional

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

Parameter	**Description**	**Required**
rtcm_dump_devname	If flag_rtcm_tty_port is set to true, descriptor of the serial port device. It defaults to /dev/pts/1.	Optional
output_path	Base path in which output data files will be stored. If the specified path does not exist, it will be created. It defaults to the current path ./.	Optional
rinex_output_path	Base path in which RINEX files will be stored. If the specified path does not exist, it will be created. It defaults to output_path.	Optional
gpx_output_path	Base path in which GPX files will be stored. If the specified path does not exist, it will be created. It defaults to output_path.	Optional
geojson_output_path	Base path in which GeoJSON files will be stored. If the specified path does not exist, it will be created. It defaults to output_path.	Optional
kml_output_path	Base path in which KML files will be stored. If the specified path does not exist, it will be created. It defaults to output_path.	Optional
xml_output_path	Base path in which XML files will be stored. If the specified path does not exist, it will be created. It defaults to output_path.	Optional
nmea_output_file_path	Base path in which NMEA messages will be stored. If the specified path does not exist, it will be created. It defaults to output_path.	Optional
rtcm_output_file_path	Base path in which RTCM binary files will be stored. If the specified path does not exist, it will be created. It defaults to output_path.	Optional

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

Parameter	**Description**	**Required**
kml_rate_ms	Output rate of the KML annotations, in ms. It defaults to the value set by output_rate.	Optional
gpx_rate_ms	Output rate of the GPX annotations, in ms. It defaults to the value set by output_rate.	Optional
geojson_rate_ms	Output rate of the GeoJSON annotations, in ms. It defaults to the value set by output_rate.	Optional
nmea_rate_ms	Output rate of the NMEA messages, in ms. It defaults to the value set by output_rate.	Optional
dump	[true, false]: If set to true, it enables the PVT internal binary data file logging. It defaults to false.	Optional
dump_filename	If dump is set to true, name of the file in which internal data will be stored. This parameter accepts either a relative or an absolute path; if there are non-existing specified folders, they will be created. It defaults to ./pvt.dat.	Optional
dump_mat	[true, false]. If dump=true, when the receiver exits it can convert the ".dat" file stored by this block into a ".mat" file directly readable from Matlab and Octave. If the receiver has processed more than a few minutes of signal, this conversion can take a long time. In systems with limited resources, you can turn off this conversion by setting this parameter to false. It defaults to true, so the ".mat" file is generated by default if dump=true.	Optional
enable_monitor	[true, false]: If set to true, the PVT real-time monitoring port is activated. This feature allows streaming the internal parameters and outputs of the PVT block to local or remote clients over UDP. The streamed data	Optional

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

Parameter	Description	Required
	members (28 in total) are the same ones that are included in the binary dump. It defaults to false.	
monitor_client_addresses	Destination IP address(es) of the real-time monitoring port. To specify multiple clients, use an underscore delimiter character (_) between addresses. As many addresses can be added as deemed necessary. Duplicate addresses are ignored. It defaults to 127.0.0.1 (localhost).	Optional
monitor_udp_port	Destination UDP port number of the real-time monitoring port. Must be within the range from 0 to 65535. Ports outside this range are treated as 0. The port number is the same for all the clients. It defaults to 1234.	Optional
enable_monitor_ephemeris	[true, false]: If set to true, the PVT real-time monitoring port streams ephemeris data to local or remote clients over UDP. It defaults to false.	Optional
monitor_ephemeris_client_addresses	Destination IP address(es) of the real-time monitoring port for ephemeris data. To specify multiple clients, use an underscore delimiter character (_) between addresses. As many addresses can be added as deemed necessary. Duplicate addresses are ignored. It defaults to 127.0.0.1 (localhost).	Optional
monitor_ephemeris_udp_port	Destination UDP port number of the real-time monitoring port for ephemeris data. Must be within the range from 0 to 65535. Ports outside this range are treated as 0. The port number is the same for all the clients. It defaults to 1234.	Optional
enable_protobuf	[true, false]: If set to true, the data serialization is done using Protocol Buffers, with the format defined at monitor_pvt.proto.	Optional

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

Parameter	Description	Required
	An example of usage is the gnss-sdr-monitor. If set to false, it uses Boost Serialization. For an example of usage of the latter, check the gnss-sdr-pvt-monitoring-client. This parameter defaults to true (Protocol Buffers is used).	
use_e6_for_pvt	[true, false]: If set to false, the PVT engine will ignore observables from Galileo E6B signals. It defaults to true, so observables will be used if found. This feature is present in GNSS-SDR v0.0.18 and later versions.	Optional
use_has_corrections	[true, false]: If set to false, the PVT engine will ignore corrections from the Galileo High Accuracy Service. It defaults to true, so corrections will be applied if available. This feature is present in GNSS-SDR v0.0.18 and later versions.	Optional
enable_pvt_kf	[true, false]: If set to true, it enables the Kalman filter of the PVT solution. It defaults to false. This configuration parameter is available starting from GNSS-SDR v0.0.19.	Optional
kf_measures_ecef_pos_sd_m	Standard deviation of the position estimations, in meters. It defaults to 1.0 [m]. Only used if PVT.enable_pvt_kf=true. This configuration parameter is available starting from GNSS-SDR v0.0.19.	Optional
kf_measures_ecef_vel_sd_ms	Standard deviation of the velocity estimations, in meters per second. It defaults to 0.1 [m/s]. Only used if PVT.enable_pvt_kf=true. This configuration parameter is available starting from GNSS-SDR v0.0.19.	Optional
kf_system_ecef_pos_sd_m	Standard deviation of the dynamic system model for position, in meters. It defaults to 2.0 [m]. Only used	Optional

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

Parameter	Description	Required
	if PVT.enable_pvt_kf=true. This configuration parameter is available starting from GNSS-SDR v0.0.19.	
kf_system_ecef_vel_sd_ms	Standard deviation of the dynamic system model for velocity, in meters per second. It defaults to 0.5 [m/s]. Only used if PVT.enable_pvt_kf=true. This configuration parameter is available starting from GNSS-SDR v0.0.19.	Optional
use_unhealthy_sats	[true, false]: If set to true, the PVT engine will use observables from satellites flagged as unhealthy in the navigation message. It defaults to false, so those observables will be ignored. This feature is present in GNSS-SDR v0.0.18 and later versions.	Optional
show_local_time_zone	[true, false]: If set to true, the time of the PVT solution displayed in the terminal is shown in the local time zone, referred to UTC. It defaults to false, so time is shown in UTC. This parameter does not affect time annotations in other output formats, which are always UTC.	Optional
rtk_trace_level	Configure the RTKLIB trace level (0: off, up to 5: max. verbosity). When set to something > 2, the RTKLIB library become more verbose in the internal logging file. It defaults to 0 (off).	Optional
bancroft_init	[true, false]: If set to false, the Bancroft initialization in the first iteration of the PVT computation is skipped. It defaults to true. This feature is present in GNSS-SDR v0.0.18 and later versions.	Optional

PVT implementation: **RTKLIB_PVT**.

Example:

```
;######### PVT CONFIG ############

PVT.implementation=RTKLIB_PVT
```

Financiado por la Unión Europea NextGenerationEU Plan de Recuperación, Transformación y Resiliencia UNICO I+D GOBIERNO DE ESPAÑA

```
PVT.positioning_mode=PPP_Static

PVT.output_rate_ms=100

PVT.display_rate_ms=500

PVT.iono_model=Broadcast

PVT.trop_model=Saastamoinen

PVT.flag_rtcm_server=true

PVT.flag_rtcm_tty_port=false

PVT.rtcm_dump_devname=/dev/pts/1

PVT.rtcm_tcp_port=2101

PVT.rtcm_MT1019_rate_ms=5000

PVT.rtcm_MT1045_rate_ms=5000

PVT.rtcm_MT1097_rate_ms=1000

PVT.rtcm_MT1077_rate_ms=1000

PVT.rinex_version=2
```

The generation of output files is controlled by the parameter output_enabled. If set to true (which is its default value), RINEX, XML, GPX, KML, GeoJSON, NMEA and binary RTCM files will be generated. You can turn off the generation of such files by setting output_enabled=false, and then select, for instance, rinex_output_enabled=true or kml_output_enabled=true. Files are stored in the path indicated in output_path, which by default is the current folder (that is, the folder from which GNSS-SDR was called). This can be changed for all outputs (for instance, output_path=gnss-products or output_path=/home/user/Documents/gnss-products/day1), or it can be defined per type of output (*e.g.*, rinex_output_path=gnss-products/rinex, gpx_output_path=gnss-products/gpx, geojson_output_path=gnss-products/geojson, etc.).

Example:

```
PVT.output_enabled=false

PVT.rtcm_output_file_enabled=false

PVT.gpx_output_enabled=true

PVT.geojson_output_enabled=true

PVT.kml_output_enabled=true

PVT.xml_output_enabled=true

PVT.rinex_output_enabled=true

PVT.nmea_output_file_enabled=false

PVT.output_path=gnss-products/others

PVT.gpx_output_path=gnss-products/gpx
```

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

```
PVT.kml_output_path=./

PVT.xml_output_path=./

PVT.rinex_output_path=gnss-products/rinex
```

This will create in your current directory:

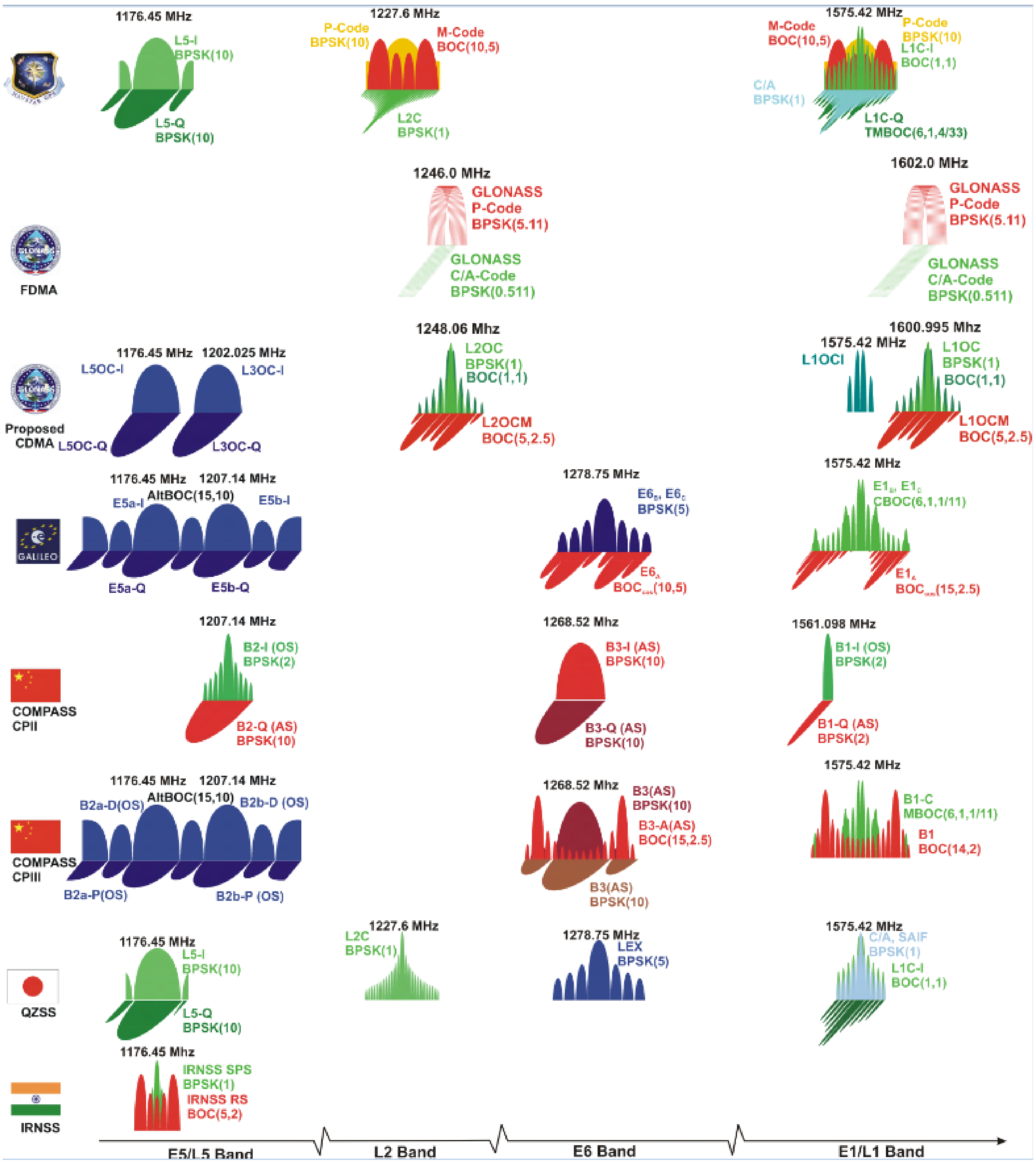
```
.
├── PVT_181028_093651.kml
├── gnss-products
│   ├── gpx
│   │   └── PVT_181028_093651.gpx
│   ├── others
│   │   └── PVT_181028_093651.geojson
│   └── rinex
│       ├── GSDR301j36.18N
│       └── GSDR301j36.18O
└── gps_ephemeris.xml
```

In order to shut down the generation of output files, you can just include in your configuration file the line:

```
PVT.output_enabled=false
```

Please note that this only concerns the generation of mentioned file formats, and it does not affect the generation of dump files activated in the configuration of each processing block. If the RTCM server is activated with flag_rtcm_server=true, it will still work even if the binary RTCM file is deactivated with rtcm_output_file_enabled=false.

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA



Source: Navipedia.

Financiado por la Unión Europea NextGenerationEU
Plan de Recuperación, Transformación y Resiliencia
UNICO I+D
GOBIERNO DE ESPAÑA

Advanced research for everyday life