elektron ámbar

Revista elektron, Vol. 8, No. 2, pp. 43-53 (2024)
https://doi.org/10.37537/rev.elektron.8.2.195.2024

Original Article

Acquisition and Processing of GPS signals based on Software Defined Radio

Adquisición y Procesamiento de señales GPS basadas en Radio Definida por Software

Castillo Delacroix L. [#1], Fagre M. [#*2], Vaquila I. [*α3], Cabrera M. A. [#4]

[#]*Laboratorio de Telecomunicaciones, Facultad de Ciencias Exactas e Ingeniería, Universidad Nacional de Tucumán*
Av. Independencia 1800, Tucumán (4000), Argentina
[1] lecastillodelacroix@herrera.unt.edu.ar
[2] mfagre@herrera.unt.edu.ar
[4] mcabrera@herrera.unt.edu.ar

[*] *Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)*
Ciudad Autónoma de Buenos Aires (C1425FQB), Argentina
[3] ivaquila@unrn.edu.ar

[α] *INVAP S. E.*
Rio Negro(R8400), Argentina

Recibido: 17/09/24; Aceptado: 02/12/24

Abstract- This paper presents the development of a software tool that implements an acquisition block for GPS signals using SDR technology, specifically devices like the HackRF One and RTL-SDR. The tool, implemented in Python, successfully detects satellites and estimates their parameters, offering customizable algorithm settings and the capability to visualize the satellite search space in three dimensions. Comparative tests with the GNSS-SDR software tool demonstrated excellent performance, providing a solid foundation for further development toward a complete SDR-based GNSS receiver. This work highlights the potential of SDR platforms for research and development, emphasizing their flexibility, potential for future upgrades, and cost-effectiveness in advancing future GNSS technologies.

Keywords: GNSS, GPS, Receiver, SDR.

Resumen- Este trabajo presenta el desarrollo de una herramienta de software que se implementa mediante un bloque de adquisición para señales GPS utilizando tecnología de SDR, específicamente dirigida a dispositivos como HackRF One y RTL-SDR. La herramienta implementada en Python detecta satélites y estima sus parámetros con muy buen desempeño, ofreciendo configuraciones de algoritmo personalizables y posee la capacidad de visualizar el espacio de búsqueda de satélites en tres dimensiones. Las pruebas comparativas con la herramienta de software GNSS-SDR demostraron un alto rendimiento, proporcionando una base sólida para expandir el desarrollo hacia un receptor GNSS completo basado en tecnología SDR. En esta propuesta se destaca el potencial de las plataformas SDR para la investigación y el desarrollo, enfatizando su flexibilidad, capacidad de actualización y rentabilidad para avanzar en futuras tecnologías GNSS.

Palabras Claves: GNSS, GPS, Receptor, SDR.

I. INTRODUCTION

In the last decades, location services through Global Navigation Satellite System (GNSS), such as the Global Positioning System (GPS), have become an essential feature of mobile devices. GPS receivers require a high level of integration, low cost, and reduced power consumption [1,2]. To achieve these requirements, they are implemented in Application Specific Integrated Circuits (ASIC) [3,4]. This solution is suitable for applications that only need to know the position of the device but limits the user's ability to interact with the architecture, algorithm, or parameters of the receiver [5,6]. Due to these limitations, the approach of GNSS Receivers implemented in Software Defined Radio (SDR) emerged, where all signal processing is done in the software domain. This approach allows modifying the algorithms or parameters in real-time or in post-processing, simply by modifying the receiver software, thus granting a high degree of flexibility to the design. The current GNSS scenario involves multi-constellation systems, which pose the challenge of designing receivers capable of processing signals of different characteristics, mitigating interference, and providing high-precision positioning, such as Precise Point Positioning (PPP). Most current civilian receivers can only decode GPS-L1 C/A signals, so research laboratories are working on receivers implemented in software [7,8,9]. Development and research on SDR-based GNSS receivers are expanding due to the flexibility and upgradeability they offer, which is crucial for advanced applications and future navigation technologies. A notable example is the open-source GNSS-SDR project of the Center Tecnològic de Telecommunications de Catalunya. This program,

developed in C++ and combined with GNU-Radio, runs on a general-purpose computer and allows selecting between different algorithms and accessing intermediate signals, which is useful to validate the development of the acquisition stage [10,11,12].

This paper presents the development of a Python-based software tool for GPS signal acquisition based on SDR devices. The acquisition stage, critical for detecting visible satellites and estimating synchronization parameters like Doppler shift and code delay, is implemented using advanced algorithms and optimized for parallel processing. Recent research has highlighted significant advancements in signal acquisition and tracking for GPS-based Software-Defined Radio (SDR) receivers, demonstrating the feasibility of using SDR platforms for efficient acquisition and tracking stages and the flexibility of SDR in optimizing signal algorithms for enhanced performance under varying environmental conditions [13,14,15]. This work contributes to the field of SDR-based GNSS receivers by providing a customizable and user-friendly platform for GPS signal acquisition. The tool was fully implemented in Python, due to the existence of libraries related to digital signal processing and the extensive documentation available [16]. Besides, it supports multiple SDR formats and incorporates visualization features to enhance usability. Comparative tests with the GNSS-SDR software validate its accuracy and performance, demonstrating its potential as a foundation for a complete SDR-based GNSS receiver capable of multi-frequency, multi-constellation processing [17,18,19, 20,21,22,23].

II. SYSTEM ARCHITECTURE

Although this development follows the SDR paradigm, the proposed diagram shown in Figure 1 is applicable to both Application Specific Integrated Circuit (ASIC) and SDR implementations. The key difference lies in the approach: in an ASIC implementation, all these blocks are hardwired using transistors integrated into a single silicon chip, making them fixed and unalterable. In contrast, the SDR approach employs a reconfigurable radio frequency interface (RF Front-End), where the SDR device is responsible for converting the analog RF signal from the antenna into digital samples. These samples are then processed by a general-purpose processor, a Digital Signal Processor (DSP), or even a Field Programmable Gate Array (FPGA). With ASICs, no block can be modified once implemented in hardware, whereas SDR implementations allow for the modification of any parameter or block in the receiver chain simply by adjusting the software. This flexibility provides a significant advantage in SDR-based implementations, making them an invaluable prototyping tool for DSP engineers, who can rapidly test and iterate different architectures or algorithms.

Figure 1. shows the GPS receiver block diagram based on SDR. The antenna used in this work was a GPS commercial patch antenna, designed for a frequency of 1575.42 MHz, corresponding to the L1 band. The RF Front-End block was implemented using an SDR device which will amplify, filter a digitize the analog signal received by the antenna. To choose the SDR hardware, we analyzed two options available in our laboratory. It's important to mention that both devices are widely used due to their low cost and good performance. The commercial HackRF One [24] is a versatile SDR platform, designed for both transmission and reception of radio signals across a wide frequency range from 1 MHz to 6 GHz. This open-source hardware device is ideal for testing, developing, and experimenting with a broad spectrum of modern and future radio technologies. Whether used as a USB peripheral or configured for standalone operation, the HackRF One offers flexibility and adaptability for a variety of applications, making it a valuable tool for hobbyists, researchers, and professionals in the field of wireless communications. The RTL-SDR [25] is a cost-effective SDR receiver, widely popular among hobbyists and professionals for its versatility and ease of use. Originating from repurposed USB TV tuner dongles, the RTL-SDR can receive frequencies from approximately 500 kHz to 1.75 GHz. It supports a wide range of applications, including radio astronomy, weather satellite image reception, ADS-B aircraft tracking, and general radio scanning. Its affordability, coupled with an extensive array of compatible software, makes the RTL-SDR an excellent entry point into the world of SDR for beginners and a valuable tool for experienced users. In Table 1, there are the main characteristics of both devices.

Tests were conducted with both devices to evaluate their reception capabilities. During the tests with the HackRF, it was impossible to achieve a lock on any satellite. After several tests, it was concluded that it needs an external clock signal because its internal oscillator is not precise enough to

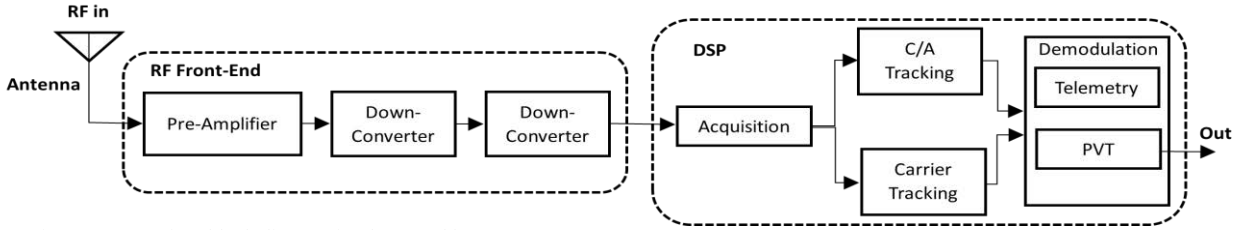

Fig. 1. GPS receiver block diagram implemented by SDR.

process GPS signals, which require 1 ppm (part per million) or less. Various software configurations were tried, even reducing the frequency and time resolution requirements to attempt to achieve a lock, but it was unsuccessful.

In the tests with the RTL-SDR dongle, excellent results were achieved for the acquisition, lock, and decoding of the navigation message, successfully obtaining the position solution.

Based on Table 1, it is observed that both devices have a suitable frequency range to capture the L1 carrier and both have an 8-bit quadrature ADC. Regarding the sampling frequency, the RTL-SDR is very close to the Nyquist sampling theorem limit, but it satisfies it, unlike the HackRF, which far exceeds it. Both have a bias-t, but only the dongle has a <1 ppm TCXO. Although the HackRF is a transceiver and the RTL-SDR is only a receiver, the application to be developed only involves reception, so this characteristic does not affect it, but the enormous price difference between the two does, precisely due to that functionality. For all the reasons mentioned above and based on the tests conducted, without making any modifications to the devices, where only the RTL-SDR achieved successful reception, we believe that the latter best suits our needs.

TABLE I.
Comparison of technical characteristics of the SDR devices analyzed

SDR	Freq. Range [MHz]	Sampling Rate [MHz]	ADC [bits]	TCXO	External Clock	Price [US$]
HackRF One	1-6000	20	8	No	Yes	250
RTL-SDRv3	24-1766	3.2	8	Yes	No	25

III. METHODOLOGY

A. Signal characteristics

The Digital Signal Processing (DSP) block shown in Figure 1 consists of three main stages: acquisition, tracking and demodulation. In this work we will focus on the acquisition stage.

Each satellite, referred to as a Satellite Vehicle (SV), currently has two unique spreading codes or sequences for signal transmission. The first one is the unencrypted Coarse Acquisition (C/A) code, assigned to civilian use, while the second is an encrypted code, known as the P(Y) code, designated exclusively for military applications and restricted from civilian access. The C/A code is modulated only on the L1 carrier frequency, whereas the P(Y) code is modulated on both the L1 and L2 carrier frequencies. In this study, our focus is on the C/A code, and its spectral characteristics are illustrated in the power spectral density diagram of the L1 band, as shown in Figure 2.

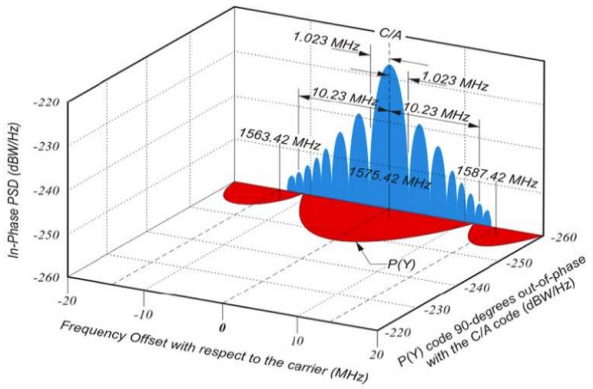

Fig. 2. Power spectral density diagram for the L1 band.[8]

The C/A code is a spread sequence that belongs to the family of Gold Codes [26]. These sequences are commonly known as pseudo-random codes or Pseudo-Random Noise (PRN) because of their apparently random nature. Pseudo-random codes are utilized because they do not repeat within their sequence, providing highly advantageous correlation properties for signal decoding. In the GPS system the C/A codes are binary and deterministic pseudo-random sequences with noise-like properties. Each code consists of 1023 chips, with a chip analogous to a bit, however, a chip does not carry useful information. These sequences consist of 512 ones and 511 zeros and repeat every 1 ms, corresponding to a frequency of 1.023 MHz.

Each SV is assigned a unique PRN code for a specific period, which may be updated over time [9]. This code is crucial because all satellites transmit simultaneously on the same carrier frequency using a Code Division Multiple Access (CDMA) scheme. In practice, this is achieved through a combination of CDMA and Direct Sequence Spread Spectrum (DSSS) technology, resulting in what is known as Direct Sequence CDMA (DS-CDMA). This encoding technique involves performing a logical Exclusive OR (XOR) operation between the PRN code and the data to be transmitted. Consequently, the signal is transmitted with a bandwidth much broader than that required by the data, reducing the power spectral density. The resulting signal exhibits a noise-like spectrum, making it undetectable and undecodable without the correct PRN code. From all this, the transmitted signal for a k satellite is given by:

$$s^k(t) = \sqrt{2P_C}\left(C^k(t) \oplus D^k(t)\right)\cos(2\pi f_{L1}t) + \sqrt{2P_{PL1}}\left(P^k(t) \oplus D^k(t)\right)\sin(2\pi f_{L1}t) + \sqrt{2P_{PL2}}\left(P^k(t) \oplus (t)\right)\sin(2\pi f_{L2}t) \quad (1)$$

where, P_C, P_{L1} and P_{L2} are the signal power of C/A and P (L1 and L2) codes, C^k and P^k are the C/A and P (Y) codes assigned to the k satellite, D^k is the navigation data, and f_{L1} and f_{L2} are the carrier frequencies L1 and L2. Since in the

present work we will only focus on the C/A code, which is modulated in the L1 band, the parameters related to the P and L2 codes are canceled. Then, our signal is simplified as follows:

$$s^k(t) = \sqrt{2P_C}\left(C^k(t) \oplus D^k(t)\right)\cos(2\pi f_{L1} t) \qquad (2)$$

When propagating from the SV to the receiver in Earth this signal is affected for different phenomena., mainly by the deviation in the L1 carrier frequency due to the Doppler effect and the propagation delay in the C/A code. The Doppler effect is the result of the frequency shift caused by the relative movement of the transmitter (located on the satellite) with respect to the receiver on the ground, even though the latter is static. The Doppler shift affects both the process acquisition and GPS signal tracking. For a stationary receiver, the maximum Doppler shift for the L1 frequency is usually considered approximately ±5 KHz, and for a moving one ±10 KHz. This effect, and the propagation delay on the C/A code in its path from the transmitter to the receiver, are two key parameters for the accurate decoding and demodulation of the signal at the receiver. The estimation of these parameters is the main function of the acquisition stage, which is the focus of our study. So, we already know the main characteristics of the transmitted signal, then we can proceed to design the acquisition stage.

B. Acquisition algorithms

The acquisition stage aims to detect the satellites visible to the receiver and obtain a first estimation of Doppler deviation and code delay. We can express the received signal as a combination of the *n* visible satellite signals:

$$s(t) = s_1(t) + s_2(t) + \cdots + s_n(t) \qquad (3)$$

Suppose we need to acquire the *i* satellite. The received signal *s* should be multiplied by a locally generated C/A code corresponding to the satellite being tracked. This ensures that only if the phase of the local code matches the phase of the received signal, meaning both codes are perfectly time-aligned, the signals from other satellites, are eliminated due to the correlation properties of the C/A codes. Similarly, the carrier frequency must be filtered, as it is affected by the Doppler shift. Therefore, knowing its exact value is necessary for proper filtering. The acquisition process can be viewed as a search for these two parameters in a two-dimensional space or grid, called the Search Grid. The axes of this grid are the Doppler Frequency and the Code Delay. Since these parameters are continuous, we must set a resolution for discretizing the space. The smallest search resolution or step is called a *bin*, which results in a frequency bin and a code bin. A combination of specific bins represents a unique position within the search space, known as a cell, as shown in Figure 3. To perform the search process, a test statistic must be evaluated in each cell. The result of this function is then compared to a predefined threshold. This comparison determines whether the satellite is present and identifies its synchronization parameters.

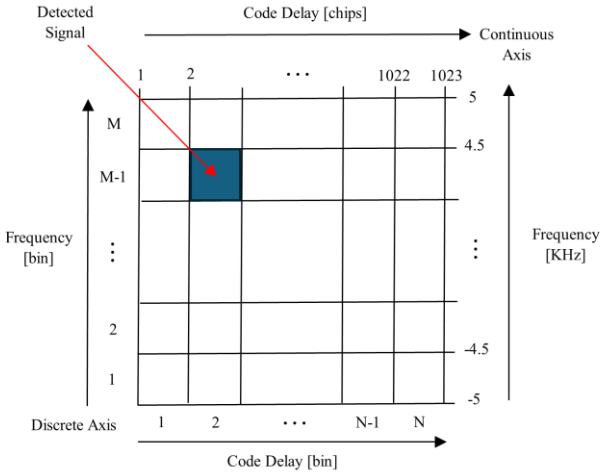

Fig. 3. Algorithm Search Space

A key aspect of any acquisition algorithm is defining the dimensions of the search space, the frequency and code search ranges, and the resolution between two consecutive bins. For the code delay axis, the range is determined by the length of the C/A code, specified by the GPS L1 C/A signal specification. This code has a length of 1023 chips, starting from position 0. The resolution between two consecutive bins must be at least one chip, resulting in 1023 code bins. However, it is possible to perform a higher-resolution search using smaller separations between bins. For the frequency axis, the range is typically ± 5 KHz for a static receiver and ±10 KHz for a moving receiver. The resolution is commonly given by frequency bins equal to *1/T*, where *T* is the coherent integration time. This implies that as the integration periods extend, the width of the frequency bins decreases, enhancing the frequency resolution. The improvements in the resolution of each axis depend exclusively on the algorithm used. There are different algorithms to perform the search, such as Serial Search (SS), Parallel Frequency Space Search (PFSS), and Parallel Code Phase Search (PCPS) acquisition, the latter being the option chosen in this work. The PCPS algorithm, whose block diagram is shown in Figure 4, performs the search by going through all possible frequency values and parallelizing the search through the code parameter, which has a significantly higher number of steps. Instead of multiplying the input signal by the local PRN for all possible code delays, as SS and PFSS algorithms do, it uses the circular cross-correlation technique based on the Fourier Transform. This technique relies on the property that the Discrete Fourier Transform (DFT) of circular cross-

correlation is equivalent to the multiplication of the DFT of the code and the DFT of the input signal, taking the complex conjugate of one of them. Performing the inverse transform of this multiplication yields the circular cross-correlation in the time domain. The squared magnitude of this result represents the test statistic and is compared with a predefined threshold to determine if the desired satellite has been acquired. If there is a peak in the test statistic, its index over the code dimension indicates the delay of the PRN code of the input signal for the analyzed frequency.

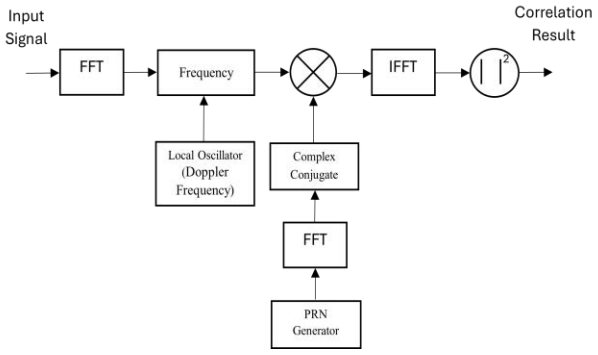

Fig. 4. Block Diagram of Parallel Code Phase Search acquisition algorithm

This algorithm sweeps only over the frequency dimension, that is, across all possible frequencies, which results in the number of combinations given by:

$$\frac{Freq.\ Range}{Freq.\ Step} + 1 = Number\ of\ combinations \quad (4)$$

The algorithm allows for defining the resolution and bin size in the frequency dimension. However, this choice is not entirely independent, as it depends on the length of the data on the DFT, which means the number of samples analyzed, and consequently the integration period. The size of a frequency bin is commonly defined as follows:

$$\Delta f = \frac{f_s}{N} = \frac{f_s}{T.f_s} = \frac{1}{T} \quad (5)$$

which results in a resolution (the maximum frequency separation between two consecutive bins) of 1/2T [12]. For example, for integration periods of 1 ms, frequency bins of 1 kHz are obtained, resulting in a maximum resolution of 500 Hz. For integration periods of 2 ms, frequency bins of 500 Hz are obtained, resulting in a maximum resolution of 250 Hz. To get better resolution, a longer integration period must be used to increase the number of DFT samples in equation 5. Additionally, the algorithm indirectly improves the resolution in the code dimension by increasing the number of samples, either by increasing the sampling frequency *fs* or the integration time *T*, as a correlation value is obtained for each sampling instant. For example, using a sample rate of 2 MHz and an integration time of 1 ms yields 2000 samples, representing 2000 bins of code instead of 1023. That is 2000/1023=1.96 samples for each bin, resulting in a resolution of 1/1.96=0.51 between bins. In summary, the accuracy of the parameters estimated with this algorithm is ±1/2 bin in frequency, and in the code dimension, it depends on the sampling frequency.

A good way to compare the algorithms mentioned above is by evaluating the number of combinations each requires and the complexity of their implementation, as shown in Table 2.

TABLE II.
Results of the comparison between SS, PFSS, and PCPS Acquisition algorithms.

Algorithm	Combinations	Complexity
Serial Search Acquisition	41943	Low
Parallel Frequency Space Search Acquisition	1023	Medium
Parallel Code Phase Search Acquisition	41	High

As observed, the Serial Search algorithm performs the worst due to the large number of combinations needed compared to the other two algorithms. The performance of both parallelized algorithms strongly depends on the implementation of the DFT. There are many implementations for the DFT, the Fast Fourier Transform (FFT) stands out as the fastest and most widely used.

As previously explained, the acquisition process involves searching in a two-dimensional grid to determine if the target satellite is present in any cell and, if so, obtaining its coordinates, which correspond to the synchronization parameters Doppler shift and code phase. This requires evaluating the test statistic in each cell (SS), column (PFSS), or row (PCPS) processed, depending on the algorithm, and comparing it with a pre-established threshold to decide if the satellite is present. In all three algorithms, the test statistics are implemented using the mathematical function "squared modulus." There are different ways to find the maximum value of the test statistic across the entire search space, which will be compared to the threshold. In the chosen implementation PCPS, the value corresponding to the maximum of the first row (or the first frequency) processed is initially stored along with its position on the grid. If the statistic for the new analyzed frequency exceeds the stored value, the new value is saved; otherwise, the process moves to the next frequency. This process continues until all frequencies are examined, resulting in the identification of the maximum test statistic across the entire search space. Calculating the decision threshold is critical for any acquisition algorithm. If the threshold value is too low, it

leads to more false positives; if too high, it may miss detecting satellites. Advanced techniques like the Constant False Alarm Rate (CFAR) [27,28] can calculate adaptive thresholds, but they add complexity and are beyond the scope of this work. Therefore, a fixed threshold of 0.019 is implemented, derived from averaging numerous tests under various propagation conditions, which can also be modified by the user to adapt to different conditions and locations. Techniques for determining satellite presence can be classified based on the number of times or windows a cell is analyzed during a non-coherent integration period:

- **Single-dwell**: Analysis is performed only once per cell to decide, using a single coherent integration window.
- **Multiple-dwell**: Analysis is repeated on the same cell at regular intervals, with averaging performed to decide, using two or more coherent integration windows that form a non-coherent integration window. The number of coherent integrations and the required positive acquisitions (max dwells) must be defined for acquisition over the non-coherent integration window.

A related parameter is the "dwell time," which is the time needed to verify the satellite presence. For single-dwell, it equals the coherent integration period; for multiple-dwell, it is the product of max dwell and the coherent integration time. Figure 5 illustrates a multiple-dwell technique with three coherent integration windows. If max dwells are set to two, at least two of the three integrations must show positive acquisition for cell acquisition. The developed tool allows users to choose between single-dwell or multiple-dwell decisions, configuring the number of dwells and max dwells accordingly. The navigation data is transmitted at 50 bps, resulting in a 20 ms period where the bit is set to 1 or -1, after which a transition may occur. If a bit of transition occurs during the acquisition process, it can cause errors. To ensure optimal algorithm performance, no bit transitions should occur in the analyzed data sequence. While longer data sequences increase the likelihood of successful satellite acquisition, they also demand more processing time and capacity. Additionally, the frequency resolution, which is inversely proportional to the integration time, improves with longer integration periods, enhancing the number of DFT samples. For all these reasons, the length of the data to be analyzed should be chosen carefully. This length should not be less than 1 ms, which is the duration of a complete C/A code. Using a shorter length would result in correlations with incomplete codes, affecting the performance of the algorithm. Therefore, the length should be an integer multiple of 1 ms. A recommended duration is 2 ms, as 1 ms does not provide good frequency resolution. However, in our implementation, this parameter can be configured by the user for each execution of the program.

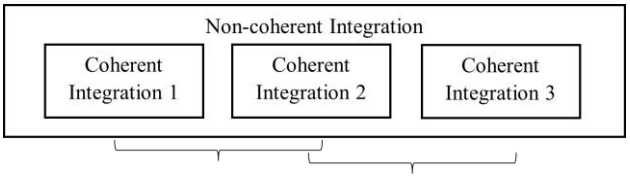

Fig. 5. Structure of an integration window with multiple dwells

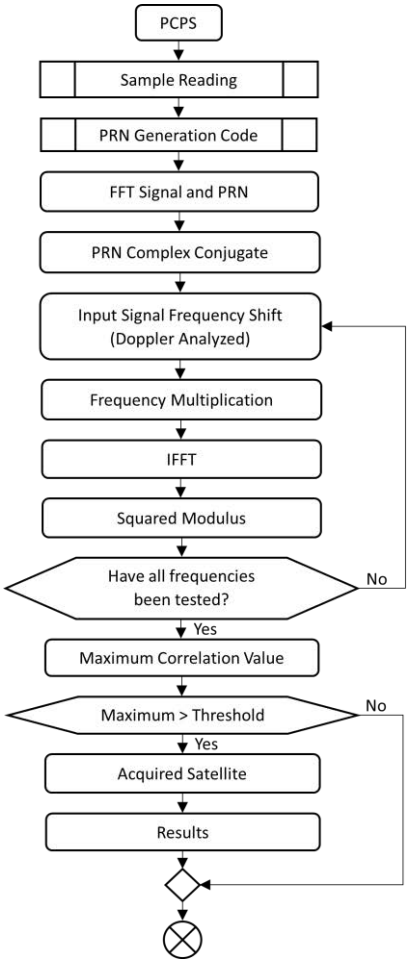

Fig. 6. Block Diagram of the initial version of the implemented algorithm

C. Implementation and Optimization

To implement the proposed algorithm, it is necessary to locally generate the PRN code of the satellite to be searched. Additionally, a small adjustment to the number of samples for each chip is required. The signal samples entering the acquisition stage have been sampled by the ADC at a specific frequency, in our case 2 MS/s. Therefore:

$$m_{chip}\,[samples] = f_s\left[\frac{samples}{s}\right] \cdot T_{chip}[s] \qquad (6)$$

$$T_{chip} = {}^{1}/_{f_{chip}} \qquad (7)$$

Considering $f_s = 2[MHz]$ and $T_{chip} = 0.977\ [\mu s]$ then from equation 6 the number of samples per chip will be $m_{chip} = 2$. It follows that for the 1023 C/A code chips, our block must generate $1023 \cdot 2 = 2046$ samples for each C/A code period, as shown below:

$$CA = \left[CA_{(1)}\ CA_{(1)}CA_{(2)}CA_{(2)} \ldots CA_{(1023)}\ CA_{(1023)}\right] \quad (8)$$

The initial version of our algorithm shown in Figure 6 was designed to search for only one satellite to validate the studied methods. The software tool was developed entirely in Python. After the first test, we observed a bottleneck in data processing, which significantly increased the processing time. To address this issue, we decided to modify the data reading process by implementing *generating functions*. These functions generate each data item only when it needs to be processed, like *lists functions*, but without keeping the data in memory. The data doesn't exist until requested, eliminating the need to load the entire list before using it. Generating functions are especially useful when dealing with large datasets, where lists can consume

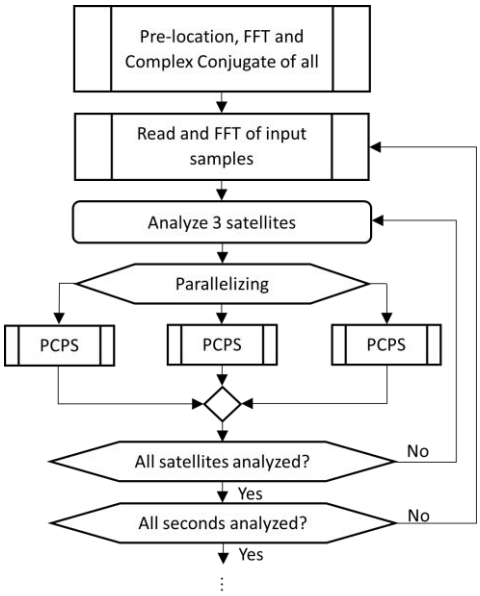

Fig. 7. Block Diagram of part of the modified version of the algorithm

significant memory, or when data is received in real-time, such as an SDR device, making it impractical to wait for all data to arrive before processing. Furthermore, a new acquisition strategy was adopted. To detect a satellite that is visible at any given second, and assuming it remains visible for at least 0.5 to 1 second, it is sufficient to search for it once, twice, or multiple times per second, depending on the number of integrations selected. It is not necessary to search for the satellite in every sample throughout all seconds of the file. Once the performance issues with the algorithm were resolved, we began the scaling process to search for all 32 satellites, our main aim. The simplest and fastest way to implement this is through a classic *for* loop, which iterates through all the satellites and applies the proposed algorithm to each one. The main disadvantage of using a *for* loop is that it was designed to search for a single satellite at a time, and in each iteration, the process of reading the input samples is repeated, even though the samples are always the same. Although it is possible to modify the reading process so that it is done only once and then the 32 satellites are searched through the loop, it was already observed in the first implementation that the processing took a considerable amount of time. Furthermore, as is characteristic of the for loop, it is a blocking execution which means that we must wait for the search for one satellite to finish before starting the next one, despite there being no limitations between individual searches for each satellite. For these reasons, and with a future implementation that can process samples in real-time in mind, we decided to investigate different parallelization methods at the processor level that allow for searching more than one satellite at the same time. This approach would also take advantage of the multicore architectures of current processors to improve performance. Currently, it is common practice to parallelize code by isolating a specific function that can be executed multiple times and running it on different processors to enable processors on a machine, by running independent parallel

TABLE III.
Comparison between different methods of Multiprocessing module in Python

Methodology	Multiple Arguments	Concurrence	Blocker	Sorted Results
Pool.map	No	Yes	Yes	Yes
Pool.map_async	No	Yes	No	Yes
Pool.apply	Yes	No	Yes	No
Pool.apply_sync	Yes	Yes	No	No
Pool.starmap	Yes	Yes	Yes	Yes
Pool.starmap_async	Yes	Yes	No	No

processes using child processes instead of threads. The maximum number of processes that can run at the same time is limited by the number of host processors. The *multiprocessing.Pool()* class provides the *apply()*, *map()*, and *starmap()* methods to execute the passed function in parallel. Choosing the most suitable one for our implementation must consider the following parameters: multiple arguments, concurrency, blocking and order of results. Table 3 presents a summary of the methods and parameters mentioned, with the aim of being able to compare them. All three methods offer synchronous and asynchronous variants. Due to the design of our algorithm, it requires a method that can handle multiple arguments and, for simplicity, able to return ordered results. According to

Table 3, the only method that satisfies these criteria is *Pool.starmap*. To implement this, we first initialize the *n* processors using the *Pool* class, then pass the functions we want to execute in parallel to *starmap*. In our case, we set the number of processors to the total number of system processors minus one, to avoid overloading the operating system and to prevent potential issues. This is a common practice that ensures efficient use of resources when running algorithms on platforms with many cores. Then, the structure of our initial algorithm is modified as shown in Figure 7.

Another modification, as shown in Figure 7, was also made regarding the way of generating the PRN codes belonging to each satellite. Previously, a single code belonging to the searched satellite had to be generated at the time of execution. But when searching for the 32 satellites, instead of generating the codes one by one each time, it is convenient to pre-allocate or previously generate a list with the codes of the 32 satellites, removing this function of the interior of the diagram proposed in Figure 6.

IV. RESULTS

To validate our implementation, we compared the obtained results with the generated by the GNSS-SDR software tool. To achieve this, the developed tool allows to record samples, and read raw sample files, whether they were recorded using this tool or with the GNSS-SDR software. Due to the existence of two different types of sample files, our reading function fits to both formats (packaging type, number of bytes per sample, sign and amplitude correction factors). Furthermore, it has been developed so that, if we need to add another type of reading format from a different SDR device, it is relatively easy to extend the code if the characteristics of the samples are known.

We chose three cases to compare these tools using GNSS-SDR format files, modifying two parameters, the coherent integration time and the Doppler resolution. The cases analyzed are using an Integration time of 1, 2 and 4 [ms] and a Doppler resolution of 1000, 500, 250 [Hz], corresponding to Case 1, 2 and 3 respectively. The results obtained from the comparison are summarized in Table 4.

As an additional verification method to identify the satellites present (excluding their synchronization parameters) detected by the developed tool, the GNSS Status application for Android devices was used while recording the signal file. This application displays real-time information on the GNSS satellites present at the location of the device, including signal-to-noise ratio, elevation and azimuth angles, position, and accuracy, among other parameters. As shown in Figure 8, the satellites detected during the file recording include satellites 2, 12, 24, 25, and 29. These results agree with those obtained by our tool and GNSS-SDR tool. Although the application lists additional satellites, their acquisition is significantly influenced by factors such as signal-to-noise ratio, line of sight, and mask angle.

Fig. 8. List of Satellites acquired by GNSS Status mobile application while recording data samples using RTL-SDR.

TABLE IV.
Comparison of the results obtained processing the GNSS-SDR input format files with GNSS-SDR software tool, and the tool developed in this work.

Case	Satellite	Software Tool			
		GNSS-SDR		This work	
		Doppler	Delay	Doppler	Delay
1	2	-	-	0	657
	12	1000	1274	1000	797
	29	-	-	2000	782
2	2	-500	1620	-500	657
	12	1000	876	1000	796
	24	-1500	1194	-1500	437
	25	2500	1900	2500	651
	29	2500	1020	2500	781
3	2	-	-	-250	657
	12	1000	649	1000	793
	24	-1500	2000	-1500	439
	25	2750	1462	2750	645
	29	2250	385	2250	775

After searching for all the satellites with the tool, it is known which of them are present, so that individual searches can be carried out for them and the graphical representation of the search space can be observed. For example, Figure 9 shows the plots obtained by our tool and by GNSSSDR, using an

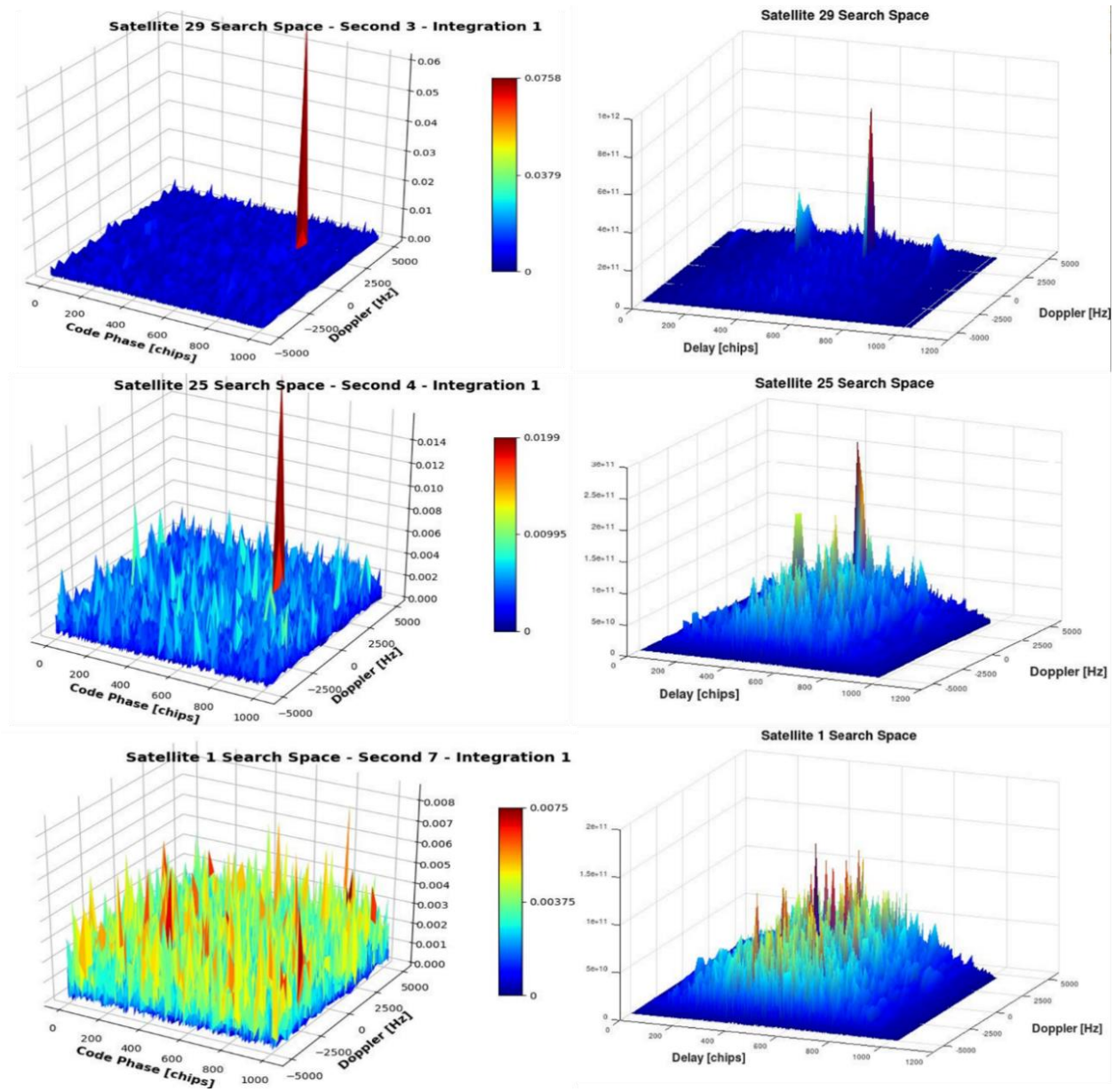

Fig. 9. Three-dimensional plots of the search space for our tool (left column) and GNSS-SDR tool (right column) for Satellite 29 (Top row), Satellite 25 (Middle row) and Satellite 1 (Bottom row).

integration Time = 2 [ms] and Doppler Resolution = 500 [Hz], for the individual searches of satellites 29 and 25 both present and satellite 1, absent. On the top and middle rows of Figure 9, corresponding to satellites 29 and 25 respectively, we can see that there is a maximum peak on the correlation matrix represented in the search space with coordinates in the Doppler frequency and delay axes that coincide with the values of these parameters previously obtained by the tool during the search for all satellites. In other words, the acquisition of these satellites has been achieved with those synchronization parameters. Also, we can see on the bottom row of Figure 9 that the same does not apply to satellite 1, which is not found as it is absent.

As shown in Table 4, each execution of the developed tool consistently produces the same code delay values for the same input file. In contrast, the GNSS-SDR software tool yields varying delay values between executions, which do not match those produced by our tool. This variability is attributed to the internal workings of GNSS-SDR, which is designed to operate in real-time across various SDR devices. It internally relies on the GNU Radio Scheduler, where each processing block runs as quickly as possible. This causes the exact execution order to fluctuate based on the machine's current load, making the process non-deterministic. However, this randomness in execution does not significantly affect the final position calculation, resulting

only in minor variations in the decimal points of longitude, latitude, and altitude. If the hardware environment is known, real-time requirements are absent, or a fixed and known amount of data is read from a file (buffer size), the sample management can be more controlled, allowing for orderly and efficient processing. This ensures a deterministic workflow, as demonstrated by the developed tool, where the same input consistently produces the same output. Additionally, the Doppler frequency results generated by the tool are identical to those obtained by GNSS-SDR in every execution.

V. CONCLUSIONS

In this work, a software tool was developed to implement the acquisition stage of GPS signals using SDR technology. The tool successfully detects satellite signals and their associated parameters, allowing users to configure various algorithm settings quickly and intuitively. Besides, it provides a graphical representation of the satellite search space in three dimensions and enables results to be saved for further analysis. The tool was fully implemented in Python, a widely used programming language in the scientific community, which simplifies maintenance and updates. Its compatibility ensures that it can run seamlessly on any computer with Python installed. The Python code used in this study is available in the GitHub repository "PyGNSS-SDR" [29]. The tool supports input data files in multiple formats, including those generated by RTL-SDR and GNSS-SDR devices, providing flexibility for different research scenarios. Validation tests demonstrated excellent agreement between the results of the developed tool and those obtained using GNSS-SDR software, confirming its accuracy and reliability.

While the tool currently represents only the acquisition block of a GNSS receiver, it establishes a robust foundation for the development of a complete receiver. The envisioned system is intended for applications such as supporting Ground-Based Augmentation Systems (GBAS), conducting ionospheric scintillation studies, and providing GPS signal integrity reports. The use of SDR technology offers a highly flexible and cost-effective approach to GNSS receiver development, particularly in the current landscape of multi-constellation systems (GPS, GLONASS, Galileo, Beidou) and multi-frequency bands. SDR platforms democratize access to GNSS research, enabling researchers and developers to engage in this field without the prohibitive costs associated with traditional hardware-based approaches. The tool developed in this project introduces a novel testing framework for GNSS research and development in our country. Future work will focus on implementing the tracking and demodulation stages, extending support to additional frequency bands (L2 and L5), and enabling compatibility with other constellations (GLONASS, Galileo, Beidou). Other planned advancements include integrating adaptive thresholding techniques, such as Constant False Alarm Rate (CFAR), for enhanced detection threshold calculation. These developments aim to further expand the capabilities and applications of the tool in GNSS research.

REFERENCES

[1] Groves, P. D., *Principles of GNSS, Inertial, and Multisensor Integrated Navigation Systems*, Artech House, 2021.

[2] Dabove, P., & Manzino, A. M., *GNSS positioning using smartphones: challenges and opportunities*, Electronics, 9(2), 258, 2020.

[3] Misra, P., & Enge, P., *Global Positioning System: Signals, Measurements, and Performance* (2nd ed.). Ganga-Jamuna Press, 2018.

[4] Kaplan, E. D., & Hegarty, C. J., *Understanding GPS/GNSS: Principles and Applications,* Artech House, 2017.

[5] Angrisano, A., Gaglione, S., & Gioia, C., *Performance assessment of assisted GNSS for smartphones in hybrid positioning mode*, Sensors, 13(9), 11485-11505, 2013.

[6] Rao, Y. S., Wang, R., & Zhang, X., *Advances in GNSS-R Technologies and Applications: A Survey*, Remote Sensing, 12(8), 1335, 2020.

[7] Petovello, M. G., & Lachapelle, G., *Software-defined GNSS receivers: Architecture, design, and future trends*, IEEE Transactions on Aerospace and Electronic Systems, 59(1), 123-138, 2023.

[8] Borre, K., & Akos, D. M., *Flexible GNSS receivers: The potential of SDR*, Navigation: Journal of the Institute of Navigation, 69(4), 321-335, 2022.

[9] Gamba, F., Tiberius, C., & Teunissen, P. J. G., Challenges and advancements in multi-constellation GNSS positioning: Focusing on Precise Point Positioning (PPP), GPS Solutions, 28(1), 23-38, 2024.

[10] Fernandez-Prades, C., & Seco-Granados, G., *GNSS-SDR: Enhancing research capabilities in GNSS signal processing with open-source tools*, Proceedings of the International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS+ 2021), 123-134, 2021.

[11] Closas, P., Seco-Granados, G., & Fernandez-Prades, C., *Open-source software-defined GNSS receivers: A versatile tool for signal processing and algorithm validation,* Navigation: Journal of the Institute of Navigation, 69(4), 407-419, 2022.

[12] Fernandez-Prades, C., Closas, P., & Seco-Granados, G., *GNSS-SDR: An open-source tool for research and experimentation in GNSS signal processing*, IEEE Transactions on Aerospace and Electronic Systems, 59(2), 789-798, 2023.

[13]Söderholm, S., Bhuiyan, M.Z.H., Thombre, S. *et al. A multi-GNSS software-defined receiver: design, implementation, and performance benefits.* Ann. Telecommun. **71**, 399–410, 2016. https://doi.org/10.1007/s12243-016-0518-7.

[14] Zhao, J., Chang, J., Yin, R., & Wang, C., *Acquisition and tracking loops based on software defined radio*, Symposium on ICT and Energy Efficiency and Workshop on Information Theory and Security (CIICT 2012), pp. 136-141, 2012. doi: 10.1049/cp.2012.1878.

[15] Htay, H., Lwin, Z., & Hla, T., *Implementation of Signal Acquisition and Tracking for GPS-Based Software Defined Radio Receiver. International Journal of Geoinformatics*, 19(2), 55–64, 2023. https://doi.org/10.52939/ijg.v19i2.2567

[16] Liu, Y., Li, J., & Wang, S., *Implementation of GNSS signal processing using Python libraries*, Digital Signal Processing, 120, 103-112, 2022.

[17] Akos, D. M., Normark, P. L., Enge, P., Hansson, A. & Rosenlind, A., *Real-Time GPS Software Radio Receiver*, Proceedings of the 2001 National Technical Meeting of The Institute of Navigation, 2001.

[18] Humphreys, T., Psiaki, M., & Kintner, P., *GNSS Receiver Implementation on a DSP: Status Challenges and Prospects*, Proceedings of the 19th International Technical Meeting of the Satellite Division of the Institute of Navigation ION GNSS, vol. 4, 2006.

[19] Borre, K., Akos, D. M., Bertelsen N., Rinder P., & Jensen S. H., *A software-defined GPS and Galileo receiver: a single-frequency approach*, Applied and Numerical Harmonic Analysis, Birkhäuser Boston, MA, 2007. https://doi.org/10.1007/978-0-8176-4540-3

[20] Schmidt, E., Akopian, D., & Pack, D. J., *Development of a Real-Time Software-Defined GPS Receiver in a LabVIEW-Based Instrumentation Environment*. IEEE Transactions on Instrumentation and Measurement, Vol. 67(9), 2082-2096, 2018.

[21] Capuano, P., Lo Presti, L., & Lohan, E. S., *Real-time GNSS signal processing using software-defined radios: Challenges and solutions*, Sensors, 22(7), 2448, 2022.

[22] Rao, B. R., & Sathyanarayana, K., *Implementation of GNSS SDR receivers: Techniques and applications*, IEEE Access, 11, 567-580, 2023.

[23] Konovaltsev, A., & Hein, G. W., *Advances in GNSS software-defined radios: From theory to practice*, Journal of Satellite Communications and Navigation, 10(2), 104-116, 2023.

[24] Great Scott Gadgets, *HackRF One [Software Defined Radio]*. https://greatscottgadgets.com/hackrf, 2014.

[25] Osmocom, *RTL-SDR [Software Defined Radio]*. https://osmocom.org/projects/rtl-sdr, 2012

[26] Gold, R., *Optimal binary sequences for spread spectrum multiplexing*, IEEE Transactions on Information Theory, 13(4), 619-621. doi:10.1109/TIT.1967.1054010, 1967.

[27] Rohling, H., *Radar CFAR thresholding in clutter and multiple target situations*. IEEE Transactions on Aerospace and Electronic Systems, AES-19(4), 608-621. doi:10.1109/TAES.1983.309362, 1983.

[28] Gao, Z., Liu, F., Wen, Y., & Wang, X., *An overview on target detection techniques in CFAR processing for non-Gaussian interference environment*. IEEE Access, 6, 5630-5645. doi:10.1P109/ACCESS.2017.2778080, 2018.

[29] Castillo Delacroix L., Fagre M., Vaquila I., Cabrera M. A., *PyGNSS-SDR (versión 1.0) [repository]*, GitHub, 2024, https://github.com/ltcfacet/PyGNSS-SDR