An Open-Source Software-Defined Receiver for GNSS Algorithms Benchmarking

Antoine Grenier, Elena Simona Lohan, Aleksandr Ometov, Jari Nurmi
Electrical Engineering Unit, *Tampere University*, Tampere, Finland
Corresponding author: antoine.grenier@tuni.fi

***Abstract*—The development of new processing algorithms has always been at the centre of satellite navigation research, aiming to minimize complexity and maximize the quality of the results. Often, these two objectives cannot be achieved simultaneously; thus, identifying the necessary trade-offs are required. Over the last decade, the development of new GNSS constellations offering modernized signals at the cost of higher decoding complexity has led to many new algorithms proposals to decode these signals. Yet, the current research environment suffers from several flaws, namely a lack of interoperability and comparability between algorithms that do not allow a fair comparison between each proposal. Such a problem is the consequence of a lack of open-source solutions, leading to individual in-house software development and dataset acquisition. To address it, we postulate the need to develop an open-source research platform providing common software and reference datasets dedicated to specific environments and user scenarios. We present the current development status and provide a discussion on future developments.**

***Index Terms*—Global Navigation Satellite System, Open source software, Computational complexity, Benchmark testing**

I. Introduction

As state-of-the-art in modern electronics, the Global Navigation Satellite Systems (GNSS) receivers have been integrated into a wide range of devices over the last decade [1]. With the growth of Internet of Things (IoT), the number of connected devices with positioning requirements is expected to rise even higher. According to [2], the number of connected IoT devices is projected to grow from 18 billion in 2022 to more than 25 billion in 2025.

The new usages of GNSS signals also lead to new challenges. Embedded devices tend to be used in difficult wireless environments where reception of GNSS signals is far from ideal: intentional/non-intentional interference, multipath, bad DOPs, etc. The design of receivers operating in these environments requires special care of these constraints to fit their user requirements. Developments in GNSS constellations also offer new signals to respond to these challenges: additional frequencies, multipath resistant codes, higher signal strength and new services [3], [4]. Modernized signals offer many advantages by design, yet their complex structures significantly impact their processing complexities [5].

A great recent effort of the research community has been to find new algorithms to improve processing efficiency. As a result, many methods have been proposed to accommodate these new signals into receivers. Yet, the advantages offered by the new methods, in terms of measurement quality or complexity reduction, can be difficult to define.

GNSS receivers in embedded devices will need to perform in an energy-constrained environment. At the 2020 GNSS User Consultation Platform, power consumption was considered a crucial parameter by 91% of users [1] and is a major challenge, as they are often pointed out as the most energy-hungry sensors [6]. Contrary to other sensors, a GNSS receiver needs to be continuously ON to produce a position. Depending on the additional information it might be able to retrieve from other channels (e.g., radio, mobile network), the receiver might require to be left on for several tens of seconds to produce a solution. While duty cycling strategies have been reviewed [7], they are delicate to implement. Offloading processing strategies, where the data is captured and transmitted to a remote server to save energy from processing on-board, have also been reviewed [8], [9]. However, positioning accuracy is greatly reduced in this configuration, and further research is needed to assess the potential energy savings.

Power consumption in electronic devices is also impacted by hardware technology evolution and electronic design [10], which are also influenced by computational requirements. The operating environment and user requirements are also major drivers for hardware design, as certain signal strength and quality performances might be needed. Understanding how these factors can influence the processing and the computational complexity is needed to select the right algorithms for an application.

Therefore, a common framework is required for an unbiased and effective comparison of algorithms' performances. In this paper, we propose the development of an open-source Software-Defined Receiver (SDR) called "pyGNSS-SDR" with multiple objectives:

1) Identification of high complexity operations in the processing chain;
2) Offering a rich and modular environment for algorithm development;
3) Benchmarking and comparing algorithm performance by a range of key metrics;
4) Testing the algorithms on reference datasets representing different environments and user scenarios;
5) Sharing the software as open-source for research and teaching purposes.

The authors gratefully acknowledge funding from European Union's Horizon 2020 Research and Innovation Programme under the Marie Skłodowska Curie grant agreement No. 956090 (APROPOS: Approximate Computing for Power and Energy Optimisation, http://www.apropos-itn.eu/).

The paper is organized as follows. In Section II, we review the motivation for a common framework and provide a survey of the existing solutions in SDRs along with their limits. In Section III, we present the detailed framework of our solution and explain its development phases. In Section IV, we review the current development status and discuss future work.

II. Motivation and major factors of impact

The prime motivation for this work is to provide answers to questions arising when designing a GNSS receiver today:

- Should the receiver be multi-constellation and multi-frequency?
- Which (if not all) signal(s) should be tracked?
- How many tracking channels should be available?
- What are the best acquisition and tracking algorithms for the target application?
- Should the device receive and use some Assisted-GNSS data (if available) to help with the receiver processing?
- What are the best algorithms & best processing design to ensure positioning reliability and integrity?
- What is the cost of all of the above in terms of hardware/software complexity?

A general answer to these questions is tough to provide. The user requirements, the operation environment, and the device resources create different constraints for each application. An "ideal" receiver is, by definition, optimized for a specific application. For energy-constrained devices, the energy required to find a positioning solution must also be assessed and will dictate which hardware/software should be implemented.

Finding the answers is crucial for a receiver's hardware and software design, and it predicts the range of applications the receiver can be suitable for. Providing such analysis is not straightforward, given the vast amount of research in algorithm design published today. Subsection II-A describes the existing challenges in more detail, while subsection II-B provides a review of existing open-source solutions and their limitations.

A. Lack of comparability

Algorithm development aggregates a significant part of the research in GNSS positioning. As GNSS receivers have applications in challenging environments and resource-constrained devices, new signals and new algorithms to acquire them are continuously published [11]–[16]. These new method proposals are often driven by improved performance at various stages of the processing chain (e.g., acquisition, tracking, navigation). The performance analysis can vary depending on the interest of the research team. While these studies always include a comparison of their solution against previous methods, it is unrealistic to expect an analysis against all possible metrics and published methods.

As a result, reviewing the performance of methods published in several independent studies is challenging as the code/reference dataset (simulated or real) is rarely provided. In the best case, it requires additional work from the consequent research teams to review previous methods. In the worst case, it can even lead to the non-repeatability of the research. Last but not least, as studies will focus on enhancing a specific part of the processing chain, the impact of the rest of the chain is often discarded. While it might be outside of a research paper's scope, understanding a method's impact on final performance metrics (e.g., position accuracy, Time To First Fix (TTFF), resource requirements) is needed to justify its usage in the process chain.

We believe the difficulties in comparing methods can be partly traced back to the lack of a shared (open-source) platform. Provided with a modular environment, new methods can be developed within an already defined platform that will discard redundant coding needs. The performance can be directly compared to previous methods under the same coding environment, using the same reference datasets and metrics. This will not ensure code sharing and will not apply to proprietary/patented techniques. Yet, such an environment could enable and encourage more open algorithm development and provide means for more fair and less biased method comparisons. Moreover, such a platform would guarantee replicability research results, as they are based on a well-defined, open environment.

Algorithm developments in GNSS processing are often performed in an SDR environment. For the sake of development efficiency, we reviewed open-source SDR solutions to base our work, which is given in the next subsection.

B. Review of open-source SDR solutions

A SDR is not a new concept, and several implementations have already been proposed. It was first introduced by [17] to simplify research algorithm development. Since then, the concept has been reused by many research teams who introduced their designs. SDRs can be separated into two categories: pure software implementations and Field-Programmable Gate Array (FPGA)-enabled implementations. The latter is preferred to typical Central Processing Unit (CPU) implementation for processing efficiency but requires programming in low-level languages. Integrated receivers are often implemented on Application-Specific Integrated Circuit (ASIC)s [23], which do not allow reconfiguration and are therefore unsuitable for algorithm testing. Instead, early-stage hardware developments use FPGA boards. On top of development modularity, FPGA boards allow the implementation of algorithms in a realistic setup, where resource requirements and power consumption can be properly estimated. Table I summarizes several SDR implementations and highlights their advantages/disadvantages. Note that for many codes defined as open-source, only a few were accessible at the time of writing. This limits the potential candidates for software enhancement.

The most well-known SDR is probably the implementation from [19], further referred to as Borre-SDR. The software was developed in Matlab and was intended as a teaching tool to understand the processing of GPS signals. It takes raw Radio Frequency (RF) samples from a recorded file as

Year	Name	Ref.	Language	Code source	Advantages	Disadvantages
1997	Akos-SDR	[17]	Matlab (CPU)	Proprietary	First implementation of a software-defined receiver.	Obsolete.
2004	Namuru	[18]	C (FPGA)	Open-source (Email access)	Real-time processing of GPS signals.	Obsolete, succeeding projects now target space-borne receiver testing.
2007	Borre-SDR	[19]	Matlab (CPU)	Open-source (Email access)	Simple implementation in high-level language.	Limited to GPS L1 C/A tracking. Architecture different from a real receiver.
2010	Macchi-SDR	[20]	Matlab (CPU)	Proprietary	Several methods for GPS and Galileo acquisition/tracking	No position computation, limited to acquisition/tracking analysis
2010	TUTGNSS	[21]	C (FPGA)	Open-source (Email access)	Real-time processing of GPS and Galileo signals.	Require development in low-level language.
2011	GNSS-SDR	[22]	C / C++ (CPU)	Open-source (Github access)	Very efficient processing on CPU, development still on-going.	Very complex code structure due to the real-time processing requirements.
2015	Guruprasad-SDR	[23]	C / C++ (FPGA)	Proprietary	Propose GPS signal processing on a low-cost platform	Require development in low-level language, limited to GPS L1 C/A tracking.
2019	BDS-SDR	[24]	Matlab (CPU)	Open-source (Link access)	Enhanced Borre-SDR capacity with BeiDou signals	Only for review of BDS decoding, not integrated as a persistent implementation.
2020	OpenCL-SDR	[25]	C++ (FPGA)	Open-source (Unaccessible)	Used OpenCL as common framework for CPU and FPGA developments	Code is open-source but unaccessible.
2022	FGI-GSRx	[26]	Matlab (CPU)	Open-source (Github access)	Large number of decoded signals	Complex code structure. Architecture different from a real receiver.
2022	Tiira	[27]	Rust (FPGA)	Open-source (Unaccessible)	Open hardware designs allowing full control of the processing	Code source not yet released.

TABLE I: Comparison of SDR published for teaching and research purposes.

input and performs all the Digital Signal Processing (DSP) operations needed before producing a position. The major strength of the software is its straightforward implementation, which is quite suitable for teaching purposes. Signal tracking is however limited to GPS L1 C/A. Recently, an upgraded version of the software called "FGI-GSRx" has been published on Github[1] by the Finnish Geodetic Institute (FGI) [26]. The software is the continuity of [19] with multi-constellation and multi-frequency capacities. Several works using the software have been published, demonstrating its usefulness in research [28], [29]. While FGI-GSRx's functionalities greatly extend upon those of Borre-SDR, its complicated implementation makes it hard to reuse and extend for our purposes.

Another popular SDR implementation is "GNSS-SDR" [22]. It is the most complete solution today, offering real-time multi-constellation and multi-frequency processing on a CPU. The software can also be used for post-processing recorded RF files. Due to its low-level language implementation in C/C++, it has been tested on several microprocessor architectures for integration in embedded electronics. The project is open-source, supported by the Centre Tecnològic de Telecomunicaciones de Catalunya (CTTC), with regular releases on their GitHub page[2]. The software has been designed to be as efficient and modular as possible. Yet, its actual implementation is complicated due to its real-time processing requirements. The core functions are based on GNU-Radio, an open-source framework for software-defined radio implementations, leading to a very efficient design but quite a complicated code structure. Moreover, coding in C/C++ at the first stages of algorithm development is cumbersome, inefficient and not easily accessible. For our purposes, the software was left for later development stages. Real-time processing capacity is unnecessary as we aim for a benchmark platform based on reference recordings. Due to its complicated structure, we decided to focus primarily on post-processing for our software development. However, the well-defined framework and design forces detailed in the GNSS-SDR documentation make it very valuable to the project.

Recently, an open hardware GNSS receiver called "Tiira" has been proposed by [27]. The goal is to provide a complete environment for algorithm developments with control over the whole processing chain. The algorithms were first developed in Python before being ported to Rust for hardware implementation. Doing so provides flexibility during the first development phase and real-time efficiency for the implementation phase. The research team developed a Proof-of-Concept (PoC) receiver for demonstration purposes along with a specifically designed RF Front-End (RFFE). A Xilinx ZYNQ board that integrates a CPU and an FPGA is used, with computationally heavy tasks like tracking algorithms efficiently integrated into the FPGA. Unfortunately, while the team announced that both

[1]See https://github.com/nlsfi/FGI-GSRx
[2]See https://github.com/gnss-sdr/gnss-sdr

software and hardware would be published as open-source in the coming future, it is not yet possible to review the implementation in-depth. Furthermore, similar to GNSS-SDR, the real-time aspect of the receiver might add unnecessary difficulties to our development.

Based on the above review, a lack of open-source software was identified for our development purposes. While some solutions exist, they either focus on real-time processing or lack modularity. For these reasons, we decided to develop our framework for algorithm development and benchmark purposes. This platform is called pyGNSS-SDR and is based on previous open-source work presented during this section.

III. A shared benchmark platform

Besides the algorithm itself, experimentation results are influenced by two factors: the data and the code implementation. Those factors can almost be considered unknown when comparing the results from different studies. In reality, if those factors are clearly outlined and defined during the experimentation phase and their access is provided in the paper, a fair comparison between any methods is ensured. Yet, those assumptions are often omitted or incomplete, which prevents the study results from being fully defined and compared. A shared environment would ensure that all the analysis is defined except for the new algorithm to be implemented, reducing the number of unknowns in the system.

A. Overall architecture

Our common benchmark platform aims to remove the burden of calibrating equipment, re-acquiring datasets and coding a complete positioning software just for the sake of algorithm development. National and international reference station networks often provide daily observations free of charge. Development at the observation level is thus blessed with a tremendous data archive. However, data recordings at I/Q-level require far greater amounts of storage, and such datasets are much harder to come by. Specific environments (e.g., open-sky, urban canyoning) and user scenarios (e.g., pedestrian, car, uncrewed aerial vehicle), required for proper algorithm benchmarking, are even rarer.

On the code side, the main objective is its modularity aspect. Essential parts of the code should be alterable, with a clearly defined Application Programming Interface (API). This is required to easily integrate new methods while having the rest of the code be independent of any changes. Fig. 1 summarizes the overall architecture. The blue parts are invariable and constant across different receiver implementations, while the green ones represent additional parts to be benchmarked.

Note that this paper will not review the reference datasets creation process, as we focus on the definition of the SDR side.

B. Software-Defined Receiver

In [30], the authors of GNSS-SDR advertised 16 design forces of an SDR. For the development of our platform, a subset of five points was selected as essentials to our scope:

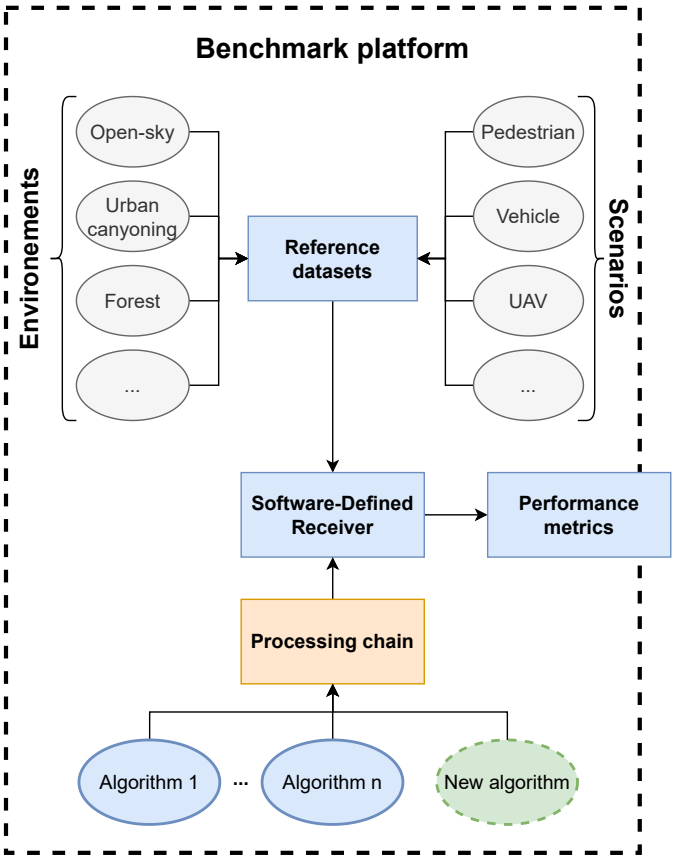

Fig. 1: Overall architecture of pyGNSS-SDR

- **Interoperability:** the system should be capable of easily incorporating new algorithms. This can be related to flexibility and/or modularity.
- **Usability:** the system should be used for different purposes (e.g., teaching, research) by individuals with different knowledge in satellite navigation (e.g., students, scientists).
- **Reproducibility:** the system should provide deterministic results given a certain code version. Proper software versioning is needed to ensure that the benchmarks can be replicated over time.
- **Openness:** the system should be provided as open-source to ensure the points described above.
- **Efficiency:** the system should have optimized resource management, as it will be part of the estimated metrics during the benchmarking.

Another essential design point is that pyGNSS-SDR is based on a concurrent state-machine [31]. Concurrent programming refers to a design where computations are executed during overlapping periods, while state-machine defines the finite number of states the receiver can be in during the processing. Contrary to the already existing high-level SDRs (e.g., Borre-SDR, FGI-GSRx), we want to mimic more closely the behaviour of a real GNSS receiver, similarly to GNSS-SDR. While this design will increase the architecture complexity

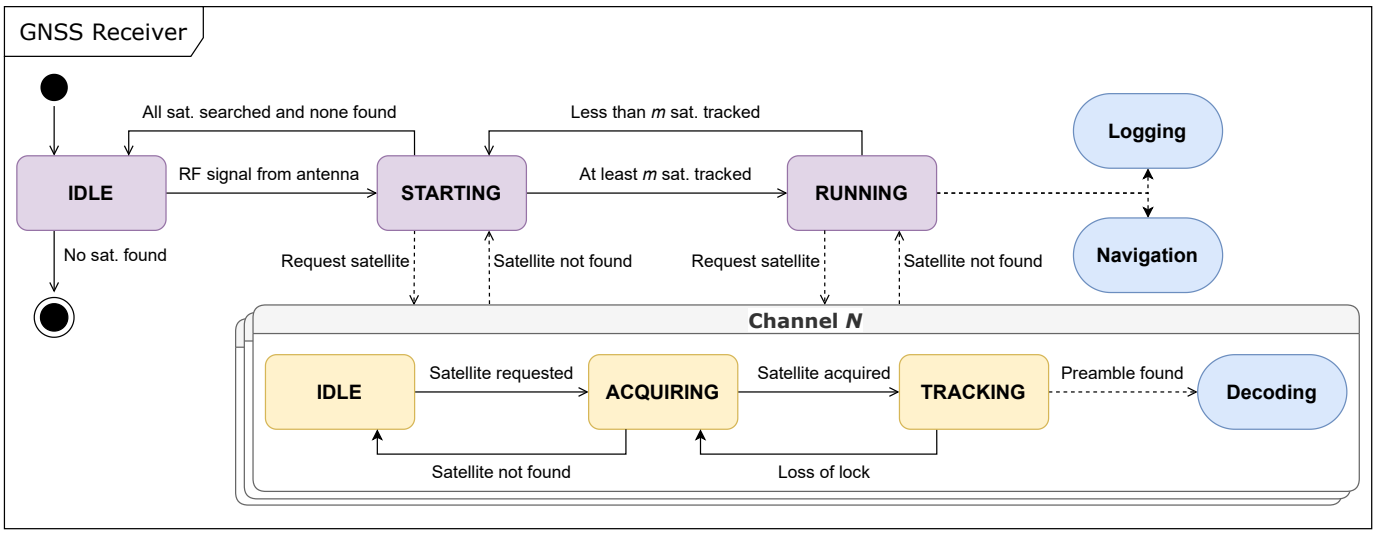

Fig. 2: State-machine diagram of pyGNSS-SDR. Purple and yellow blocks represent the receiver and channel states, respectively. Blue blocks are additional flags enabled by switching to different states.

of the receiver, it should provide more realistic end-to-end processing without preventing any benchmark operations. As a low-level implementation is already foreseen, adopting a design as close as possible to an actual receiver should simplify the future development of an embedded version of the receiver.

The state-machine diagram of the receiver is presented in Fig. 2. The design is defined as concurrent, as the receiver can be segmented into two main operations: channel tracking and navigation. At the receiver's start, N channels are created depending on the receiver parameterization. Specific satellites are requested to be acquired and tracked by these channels, which then progress/proceed in an "independent" manner. The transition between acquisition and track depends on each algorithm's decision threshold. Meanwhile, the main receiver loop monitors all the channels' status as the processing continues. Once a certain number of satellites are being tracked, the receiver will switch to a running mode, where it will try to perform navigation.

Depending on the scenario and the assisted data available to the receiver, this might require partial/full decoding of the navigation message within the GNSS signals. While this is a simple and standard architecture, it allows us to realistically simulate the state a GNSS receiver passes through during the processing. It is essential to define the relations between the different processing steps and obtain a comprehensive behaviour trace of the algorithms once they are implemented together. Moreover, the division of processes allows us to define the interface required between algorithms more clearly.

C. Development phases

Due to the complexity of creating an SDR from scratch, the development of the receiver is foreseen in three different phases, differentiated by the programming level of the SDR. On the one hand, low-level programming offers better efficiency but will lead to longer development time due to the inherent coding difficulty [17]. Moreover, it will reduce the interoperability and usability of the software, as low-level programming knowledge would be needed to extend its capabilities. On the other, high-level programming offers design flexibility and focuses on algorithm design, but at the price of reduced efficiency. High-level languages often offer the possibility of wrapping and calling functions written in lower-level languages to increase efficiency. This study considers compiled languages like C/C++/Rust as low-level languages compared to Matlab/Python.

The first phase is the development of a pure software SDR, coded in a high-level language. It is similar to Borre-SDR and FGI-GSRx software, yet with fundamental design differences.

1) Phase I: Our first objective is to create an entirely virtual replica of a GNSS receiver. The platform designed will allow high-level algorithm development for testing and debugging purposes. It shall provide performance metrics for tracking/positioning quality and code efficiency. High-level programming will prevent realistic resource consumption estimation but should give an overall idea of the algorithm's efficiency. Following our focus on open code, we decided to write the platform in Python. While Matlab is used widely for its readability and easy programming development, Python offers similar advantages without requiring any licensing. Moreover, the current trends in programming have rendered Python quite versatile and efficient, leading to broad usage in research and industry, as seen, for example, in Tiira [27]. Python can also easily integrate low-level languages, which is interesting for the second development phase.

2) Phase II: Going from the fully high-level software, a conversion to lower-level languages is already foreseen. The goal is to keep the backbone of the receiver in a high-level language structure while partially mixing the code with more optimized languages. Compiled languages similar to C/C++

are reviewed for this phase's development. In [27], Rust was selected to develop the Tiira receiver's low-level part. It also answers our requirements on open code and aligns with current coding trends. This partial/total conversion to low-level languages leads to several advantages: (1) optimization of highly demanding computational tasks (e.g., DSP); (2) better computational resource management and monitoring; and (3) code encapsulation, i.e., isolation of specific tasks into other languages. The third point is particularly important to ensure the interoperability of our software. Moreover, it paves the way for the third phase.

3) Phase III: Proper estimation of resources and algorithm energy consumption requires dedicated hardware. In the previous phases, the code runs on a CPU in a typical operating system environment. In the pursuit of energy monitoring and realistic simulation of the receiver behaviour, the code should be isolated on specific hardware, such as an FPGA board. Energy consumption translation from FPGA to ASIC hardware is possible, as shown in [32]. Such a solution was chosen for implementing the Tiira receiver [27], using both a CPU and FPGA processor unit in the Xilinx ZYNQ board. A similar implementation is foreseen for this development phase on both simulated and physical hardware.

We have reviewed the development phases planned for pyGNSS-SDR. As its development continues, this preliminary plan is subject to change and will adapt to unforeseen challenges. Moreover, the different phases have been axed around the programming level, starting with a pure software environment and moving toward hardware implementation. Due to research purposes, linear development is impossible, and parallel work at multiple programming levels will probably happen. Similar capacities across the software versions are required to ease future algorithm developments. Thus, parallel development of the pure software receiver and hardware implementation will be performed to ensure the continuity and interoperability of both versions.

IV. Intermediate Results

Development of Phase I is ongoing. The receiver can perform acquisition, tracking, and decoding operations of multiple GPS L1 C/A signals on different virtual channels. The methods implemented are based on literature reviews [3], [19] and a study of the various open-source SDR reviewed in Section II.

A preliminary overview of the current software capacities can be found in Figs. 3 and 4. The acquisition and tracking results are presented for the GPS L1 C/A signal of satellite PRN 2. Details on the experimental setup, algorithms and parameters used to produce these results are given in Table II. Note that the parameters were set based on the configuration provided in the other SDRs and numbers found in literature [3], [19]. They are not optimized for performance or efficiency purposes.

The first release will propose an end-to-end processing chain, from I/Q files to standard positioning, with minimum performance metrics analysis. For our research purposes, we

Reference dataset		
Date		2021.11.30, $\sim$ 8:40 (UTC)
Location		TAU Rooftop (open-sky)
Dynamic		Static
Antenna		Novatel GPS-703-GGG
Instruments	RF Logger	NI USRP-2953R
	Clock	Spectracom GSG-6
Frequency	Center	1575.42 MHz
	Bandwidth	120 MHz
	Sampling frequency	10 MHz
	Quantization	8 bits
	I/Q	Complex
Acquisition		
Method		PCPS
Doppler Range		$\pm$ 5000 MHz, 100 MHz step
Integration		5 ms coherent, 10 ms non-coherent
Threshold		Ratio two highest peak, 1.5
Tracking		
Method		Early Prompt Late
Correlator spacing		-0.5 / 0 / 0.5
DLL	PDI	0.001
	Dumping ratio	0.7
	Noise bandwidth	2.0 Hz
	Loop gain	1.0
PLL	PDI	0.001
	Dumping ratio	0.7
	Noise bandwidth	25.0 Hz
	Loop gain	0.25

TABLE II: Simulation parameters of the preliminary results.

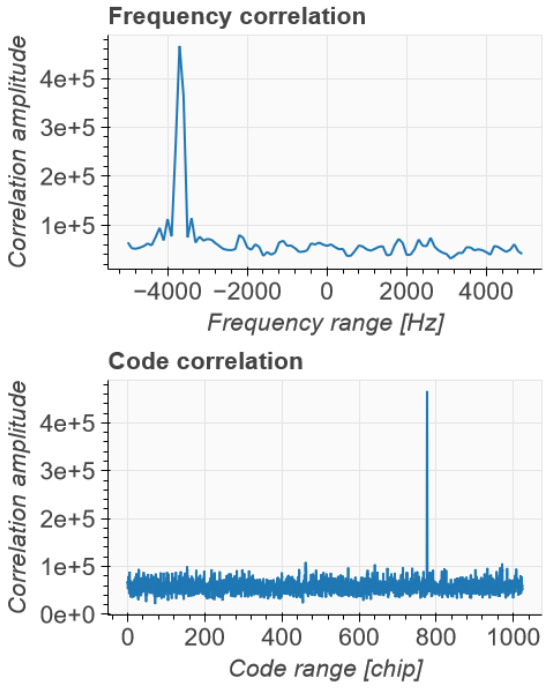

Fig. 3: Preliminary acquisition results for GPS L1 C/A signal of the satellite PRN 2.

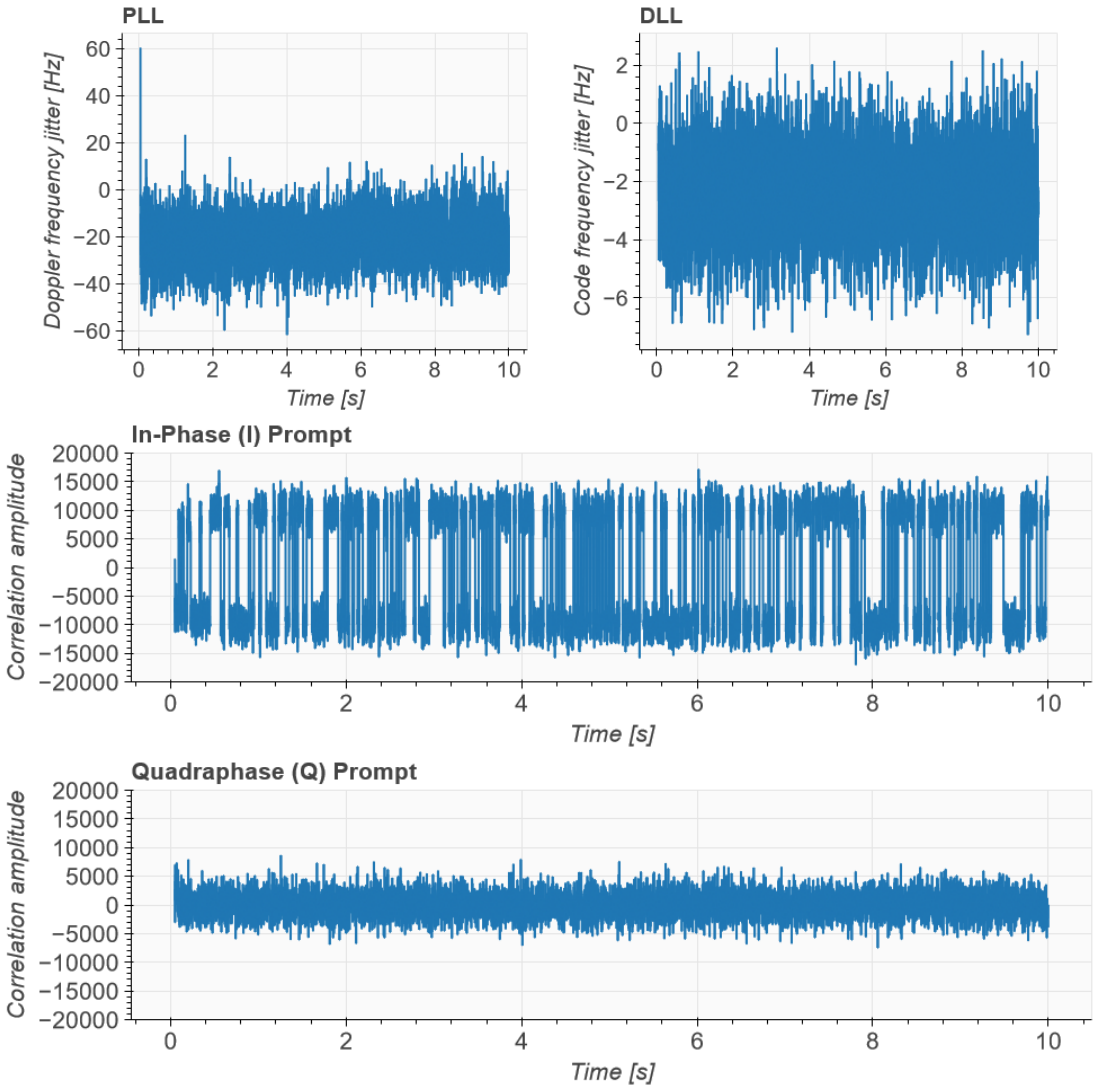

Fig. 4: Preliminary tracking results for GPS L1 C/A signal of the satellite PRN 2.

initially focus on enabling GPS L1 C/A in the SDR, providing a mono-constellation, mono-frequency solution. It will constitute our baseline for the benchmarks realized with the software. The publication of this first code version is foreseen for the end of the year. Additional performance metrics and actual positioning results will be further reviewed during the conference presentation, and a link to our GitHub page will be provided.

Subsequent updates and releases will gradually enhance the receiver capacities to integrate multi-frequencies (e.g. L5) and additional constellations, e.g., Galileo and BeiDou.

V. Conclusion

This paper highlights the growing number of algorithms for DSP of modern GNSS signals. The methods are often compared to a limited number of similar methods, while neither the code nor the reference dataset is shared. This leads to difficulties in comparing algorithm performances. We reviewed the existing solutions in open-source SDR software to resolve this issue.

While several options exist, none constitute a clear, modular and easy-to-develop environment that would enable algorithm comparability. Thus, we proposed the development of an open-source platform containing both reference datasets and codes to perform algorithm benchmarking. We focused on defining the SDR coding architecture and explained the foreseen development phases of the tool. We highlighted the design forces of our tool related to the difficulties we identified in GNSS algorithm research. Finally, the current status of the receiver has been presented, along with a description of the first release objectives and the challenges linked to its development.

REFERENCES

[1] European Union Agency for the Space Programme (EUSPA), "Market report," 2022. Issue 1.

[2] European GNSS Agency (GSA), "Power-Efficient Positioning for the Internet of Things," 2020. White Paper.

[3] E. D. Kaplan and C. J. Hegarty, *Understanding GPS, Principles and Applications. 3rd Edition*. Artech House, 2017.

[4] I. Fernandez-Hernandez, A. Chamorro-Moreno, S. Cancela-Diaz, J. Calle-Calle, P. Zoccarato, D. Blonski, T. Senni, F. Blas, C. Hernández, J. Simón, and A. Mozo, "Galileo High Accuracy Service: Initial definition and Performance," *GPS Solutions*, vol. 26, 07 2022.

[5] J. Leclère, R. Landry, and C. Botteron, "Comparison of L1 and L5 Bands GNSS signals acquisition," *Sensors*, vol. 18, p. 2779, 08 2018.

[6] S. Narayana, R. V. Prasad, V. Rao, L. Mottola, and T. V. Prabhakar, "Hummingbird: Energy Efficient GPS Receiver for Small Satellites," in *Proc. of the 26th Annual International Conference on Mobile Computing and Networking*, MobiCom '20, (New York, NY, USA), Association for Computing Machinery, 2020.

[7] V. Bellad, *Intermittent GNSS Signal Tracking for Improved Receiver Power Performance*. PhD thesis, University of Calgary, 2015.

[8] H. Ramos, T. Zhang, J. Liu, N. Priyantha, and A. Kansal, "LEAP: A low energy assisted GPS for trajectory-based services," in *UbiComp'11 - Proc. of the ACM Conference on Ubiquitous Computing*, pp. 335–344, 09 2011.

[9] P. Misra, W. Hu, Y. Jin, J. Liu, A. S. de Paula, N. Wirström, and T. Voigt, "Energy efficient GPS acquisition with Sparse-GPS," in *IPSN-14 Proc. of the 13th International Symposium on Information Processing in Sensor Networks*, pp. 155–166, 2014.

[10] B. Aebischer and L. Hilty, *The Energy Demand of ICT: A Historical Perspective and Current Methodological Challenges*, vol. 310, pp. 71–103. Springer, 08 2015.

[11] J. Leclère, C. Botteron, R. J. Landry, and P.-A. Farine, "FFT splitting for improved FPGA-based acquisition of GNSS signals," *International Journal of Navigation and Observation*, vol. 2015, p. 12, 12 2015.

[12] T. Feng, "Decimation Double-Phase Estimator: An Efficient and Unambiguous High-Order Binary Offset Carrier Tracking Algorithm," *IEEE Signal Processing Letters*, vol. 23, no. 7, pp. 905–909, 2016.

[13] T. Li, Z. Tang, J. Wei, Z. Zhou, and B. Wang, "An Unambiguous Tracking Technique for Cosine-Phased BOC Signals with Low Complexity," *Radio Engineering*, vol. 27, pp. 1191–1198, 09 2018.

[14] B. Wang, T. Li, J. Wei, and Z. Tang, "A new unambiguous tracking algorithm for sine-BOC(m, n) signals," *GPS Solutions*, vol. 23, 04 2019.

[15] J. Svatoň, F. Vejražka, P. Kubalík, J. Schmidt, and J. Borecký, "Novel partial correlation method algorithm for acquisition of GNSS tiered signals," *NAVIGATION*, vol. 67, no. 4, pp. 745–762, 2020.

[16] F. Hao, B. Yu, X. Gan, R. Jia, H. Zhang, L. Huang, and B. Wang, "Unambiguous Acquisition/Tracking Technique Based on Sub-Correlation Functions for GNSS Sine-BOC Signals," *Sensors*, vol. 20, no. 2, 2020.

[17] D. M. Akos, *A software radio approach to global navigation satellite system receiver design*. Ohio University, 1997.

[18] F. Engel, P. Mumford, K. Parkinson, C. Rizos, and G. Heiser, "An open GNSS receiver platform architecture," *Journal of Global Positioning Systems*, vol. 3, pp. 63–69, 12 2004.

[19] K. Borre, D. M. Akos, N. Bertelsen, P. Rinder, and S. H. Jensen, *A Software-Defined GPS and Galileo Receiver, a single frequency approach*. Birkhäuser, 2007.

[20] F. Macchi, *Development and testing of an L1 combined GPS-Galileo software receiver*. PhD thesis, University of Calgary, 2010.

[21] T. Paakki, J. Raasakka, F. Della Rosa, H. Hurskainen, and J. Nurmi, "TUTGNSS University based hardware/software GNSS receiver for research purposes," in *2010 Ubiquitous Positioning Indoor Navigation and Location Based Service*, pp. 1–6, 2010.

[22] C. Fernández–Prades, J. Arribas, P. Closas, C. Avilés, and L. Esteve, "GNSS-SDR: An open source tool for researchers and developers," in *Proc. 24th Int. Tech. Meeting Sat. Div. Inst. Navig.*, (Portland, OR), pp. 780–794, Sept. 2011.

[23] S. Guruprasad, *FPGA-Based Software GNSS Receiver Design for Satellite Applications*. PhD thesis, York University, 2015.

[24] Y. Li, N. C. Shivaramaiah, and D. M. Akos, "Design and implementation of an open-source BDS-3 B1C/B2a SDR receiver," *GPS Solutions*, vol. 23, p. 60, Apr 2019.

[25] J. Buttgereit, T. Schwarte, and G. C. Kappen, "Design and Implementation of a Software Defined Radio GNSS Receiver Based on OpenCL," in *Proc. of IEEE/ION Position, Location and Navigation Symposium (PLANS)*, pp. 1237–1246, 2020.

[26] Finnish Geodetic Institute, "The FGI-GSRx software defined GNSS receiver goes open source," 2022. Published: 2022-02-09.

[27] O. Daniel, M. Pfleger, V. Jenik, V. Talyzin, O. Kost, and J. Dunik, "Tiira: an Open-Source Hardware-based GNSS Receiver and Multi-sensor Navigation System," in *NAVITEC*, 2022.

[28] S. Söderholm, M. Z. H. Bhuiyan, S. Thombre, L. Ruotsalainen, and H. Kuusniemi, "A multi-GNSS software-defined receiver: design, implementation, and performance benefits," *Annals of Telecommunications*, vol. 71, 05 2016.

[29] N. Linty, M. Z. H. Bhuiyan, and M. Kirkko-Jaakkola, "Opportunities and Challenges of Galileo E5 Wideband real signals processing," in *Proc. of International Conference on Localization and GNSS (ICL-GNSS)*, pp. 1–6, 2020.

[30] C. Fernández–Prades, J. Arribas, and P. Closas, "Assessment of Software-Defined GNSS receivers," in *NAVITEC*, Dec. 2016.

[31] E. A. Lee, "Concurrent Models of Computation: concurrent state machines," 2011.

[32] I. Kuon and J. Rose, "Measuring the Gap Between FPGAs and ASICs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007.