An Open Source Galileo E1 Software Receiver

Carles Fernández–Prades*, Javier Arribas*, Luis Esteve†, David Pubill*, and Pau Closas*
*Centre Tecnològic de Telecomunicacions de Catalunya (CTTC)
Av. Carl Friedrich Gauss 7, 08860 Castelldefels, Spain. Email: {carles.fernandez, javier.arribas, david.pubill, pau.closas}@cttc.cat
†Universitat Politècnica de Catalunya (UPC), Dept. of Signal Theory and Communications
Jordi Girona 1–3, 08034 Barcelona, Spain. Email: luis@epsilon-formacion.com

***Abstract*—This paper presents an open source implementation of a GNSS software receiver that targets Galileo E1B and E1C signals. After discussing the signal structure and the general architecture of the proposed software receiver, we provide detailed descriptions of the main signal processing algorithms involved in acquisition and tracking of such navigation signals. Experimental results with signals broadcast by in-orbit Galileo satellites validate the approach. The connection to external software (that allows in-the-loop testing of algorithms implemented in popular commercial tools) is also presented and validated with real-life signals.**

I. Introduction

The race to complete the Galileo satellite constellation has finally started. Two In-Orbit-Validation (IOV) satellites are transmitting navigation signals since December 12, 2011, and two more IOV satellites were launched by October 12, 2012. An initial service constellation of 18 satellites is expected by late 2014, and by the end of the decade a fully operational service with a minimum of 30 satellites will be available, according to the European GNSS Supervisory Authority (GSA).

As a matter of fact, the landscape of GNSS is going to change rapidly in the following years (modernization of GPS and GLONASS, advent of Galileo and COMPASS). A number of new signals will be readily available for navigation, providing means to determine position and time with an unforeseen degree of performance. Nevertheless, the multi-constellation, multi-frequency approach poses several technological challenges. In that sense, the flexibility provided by the software defined radio approach appears as an ideal environment for rapid prototyping and testing of new receiver architectures and algorithms.

In response to this growing demand, in previous publications [1], [2] we presented an open-source GNSS software defined receiver (so-called GNSS-SDR, available at http://gnss-sdr.org) that makes use of the well-known GNU Radio framework [3]. It provides a working implementation of a whole processing chain of a GPS L1 C/A receiver, from the output of a RF front-end to the computation of position, velocity and time. It also outputs standard formats (KML and NMEA for graphical representation, and RINEX for observables and navigation data).

In the present work, we extend the receiver functionality to acquire, track, and demodulate the navigation message of the Galileo E1 open signal, including both E1B data and E1C pilot components. Next Section describes the mathematical model of the target signals, and then Section III presents the proposed software receiver architecture and the signal processing chain. Section IV shows some experimental results, and finally Section V concludes the paper.

II. Signal model

The analytic representation of a signal received from a generic GNSS satellite can be expressed as

$$r(t) = a(t)s_T\left(t-\tau(t)\right)e^{-j2\pi f_d(t)}e^{j2\pi f_c t} + n(t)\ , \qquad (1)$$

where $a(t)$ is the complex amplitude, $s_T(t)$ is the complex baseband transmitted signal, $\tau(t)$ is the time–varying delay, $f_d(t) = f_c\tau(t)$ is the Doppler shift, f_c is the carrier frequency, and $n(t)$ is a noise term. The waveform $s_T(t)$ and its parameters depend on the particular GNSS and frequency band used by the receiver. The Galileo E1 band, centered at $f_c^{(Gal\ E1)} = 1575.420$ MHz and with a reference bandwidth of 24.5520 MHz, uses the so-called composite binary offset carrier CBOC(6,1,$\frac{1}{11}$) modulation, defined in baseband as:

$$\begin{aligned} s_T^{(Gal\ E1)}(t) = &\frac{1}{\sqrt{2}}\Big(e_{E1B}(t)\left(\alpha sc_A(t)+\beta sc_B(t)\right)+ \\ &- e_{E1C}(t)\left(\alpha sc_A(t)-\beta sc_B(t)\right)\Big)\ , \end{aligned} \qquad (2)$$

where the subcarriers $sc_A(t)$ and $sc_B(t)$ are defined as

$$sc_A(t) = \mathrm{sign}\Big(\sin(2\pi f_{s,E1A}t)\Big)\ , \qquad (3)$$

$$sc_B(t) = \mathrm{sign}\Big(\sin(2\pi f_{s,E1B}t)\Big)\ , \qquad (4)$$

and $f_{s,E1A} = 1.023$ MHz, $f_{s,E1B} = 6.138$ MHz are the subcarrier rates, $\alpha = \sqrt{\frac{10}{11}}$, and $\beta = \sqrt{\frac{1}{11}}$. The E1B component contains the bit sequence of the Integrity Navigation Message, $D_{\text{I/NAV}}$, transmitted at 250 sps and intended for navigation, integrity and Safety–of–Life (SoL) services:

$$e_{E1B}(t) = \sum_{l=-\infty}^{+\infty} D_{\text{I/NAV}}\Big[[l]_{4092}\Big] \oplus C_{E1B}\Big[|l|_{4092}\Big] p(t - lT_{c,E1B}), \qquad (5)$$

where $\oplus$ is the exclusive–or operation (modulo–2 addition), C_{E1B} is a pseudorandom noise (PRN) code (unique for each satellite), $[l]_L$ means the integer part of $\frac{l}{L}$, $|l|_L$ means l modulo L, $T_{c,E1B} = \frac{1}{1.023}$ μs, and $p(t)$ is a rectangular pulse of

a chip–period duration centered at $t = 0$ and filtered at the transmitter.

The E1C component is a pilot (dataless) signal with a secondary code, forming a tiered code:

$$e_{E1C}(t) = \sum_{m=-\infty}^{+\infty} C_{E1Cs}\Big[|m|_{25}\Big] \oplus \sum_{l=1}^{4092} C_{E1Cp}\Big[l\Big] \cdot \cdot p(t - mT_{c,E1Cs} - lT_{c,E1Cp}) \ , \quad (6)$$

with $T_{c,E1Cp} = \frac{1}{1.023}$ μs and $T_{c,E1Cs} = 4$ ms. The C_{E1B} and C_{E1Cp} primary codes are pseudorandom memory code sequences defined in [4, Annex C.7 and C.8]. The binary sequence of the secondary code C_{E1Cs} is 001110000000010101110110010. This band also contains another component, Galileo E1A, intended for the Public Regulated Service (PRS), that uses a so-called BOC(15,2.5) modulation with cosine–shaped subcarrier $f_{s,E1A} = 15.345$ MHz and $T_{c,E1A} = \frac{1}{2.5575}$ μs. The PRS spreading codes and the structure of the navigation message have not been made public, and hence E1A is not a target of the present work.

As is the case of other GNSS signals, the receiving power level at the Earth surface of $r(t)$ is extremely weak, well below the noise floor. The minimum received power on ground, defined at the output of an ideally matched right-hand circularly polarized 0 dBi user receiving antenna when the satellite elevation angle is higher than 10 degrees, is -157 dBW, considering 50/50 % E1B/E1C power sharing.

III. Software Receiver Implementation

In order to acquire, track and demodulate those signals, we based on GNSS-SDR, an open source software receiver written in C++. The general software architecture was described in [2], and it is sketched in Figure 1. It currently allows the use of several radio frequency front-ends, and it can also work off-line with raw signal samples stored in a file. Users can define a custom signal processing flowgraph (type of signal source, number of channels, algorithms to be used for each channel and each module, strategies for satellite selection, type of output format, etc.) through a configuration file, and populate the signal processing blocks with their own algorithms.

The flowgraph is managed by a Control Plane, a thread that runs concurrently with the processing blocks and is in charge of receiving notifications and triggering run-time changes in the application. Some of these notifications will be sent directly from the processing blocks. For instance, an Acquisition block that detects the presence of a satellite's signal will send a notification to the control thread via a message queue, indicating its success. The control thread will then change the internal configuration of the channel and pass the results of the acquisition process to the Tracking block.

For the present work, we developed specific Galileo E1 implementations for the blocks labeled as *Signal Source*, *Signal Conditioner*, *Acquisition*, *Tracking* and *Telemetry Decoder* in Figure 1, as well as the corresponding addition of Galileo satellites in the control plane (activating PRN 11 and 12, the two in-orbit satellites at that time), and a number of unit tests that ensure the correct functionality of such implementations.

A. Signal source

The input of a software receiver are the raw bits that come out from the front-end's analog-to-digital converter (ADC). 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. The *Signal Source* block is an abstraction layer that hides the complexity of accessing each specific signal source, providing a single interface to a variety of different implementations. Depending on how the signal samples are obtained, the user can configure the receiver for reading from a file (and thus specifying the file location, sample format, and the sampling frequency and intermediate frequency at what the signal was originally captured) or for interacting with a hardware front-end, if the host processor is fast enough to allow real-time processing. Currently, GNSS-SDR provides interfaces to the family of Ettus Research products through the Universal Software Radio Peripheral (USRP) Hardware Driver (UHD), allowing to set specific front-end parameters such as the center frequency and the front-end gain; to the SiGe GN3S Sampler v2 USB front-end, originally intended to capture data for the Matlab-based GNSS software receiver available from K. Borre's book [5]; and to some low-cost USB DVB-T dongles based on the Realtek RTL2832U chipset. The user can also write custom drivers for any other hardware device and wrap them as Signal Sources, allowing to expand the functionality of the software receiver to potentially any other RF front-end.

At the block output, there is an amplified, downconverted, decimated, filtered and digitized version of $r(t)$, becoming a signal stream source that injects samples into the flowgraph at the ADC sampling rate. Following the diagram of Figure 1, the Signal Source block output feeds the input of a Signal Conditioner.

B. 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 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 $x_{\text{IN}}[n]$. This signal stream feeds a set of parallel *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 and hyperthreading processors. In the following, we discuss

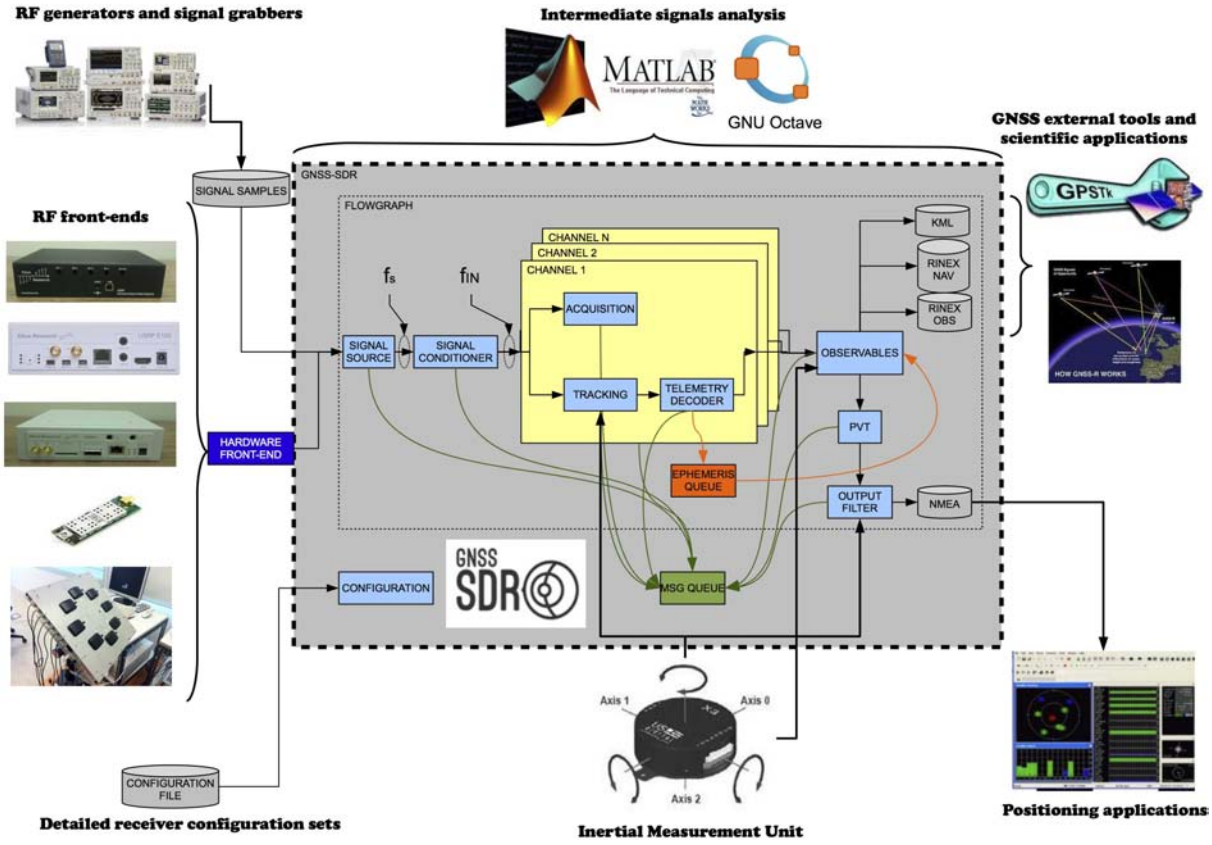

Fig. 1. Overview of the proposed GNSS software receiver. The user can define the signal source, the flowgraph and the specific algorithms and parameters for each of the processing blocks through the configuration file. Signal can be injected or dumped into a file at any stage of the processing chain, allowing the interaction with a wide range of external data and analysis tools.

implementation details of those blocks specifically addressed to Galileo E1B and E1C signals.

C. Acquisition

The role of an *Acquisition* block is the detection of presence/absence of signals coming from a given GNSS satellite. In case of a positive detection, it should provide coarse estimations of the code phase $\hat{\tau}$ and the Doppler shift $\hat{f}_d$, yet accurate enough to initialize the delay and phase tracking loops. By exploiting the concepts and the methodology of the estimation theory, it is possible to show that the maximum likelihood (ML) estimates of f_d and τ can be obtained by maximizing the function

$$\hat{f}_{d_{ML}}, \hat{\tau}_{ML} = \arg\max_{f_d,\tau} \left\{ \left| \hat{R}_{xd}(f_d,\tau) \right|^2 \right\} \ , \qquad (7)$$

where

$$\hat{R}_{xd}(f_d,\tau) = \frac{1}{N} \sum_{n=0}^{N-1} x_{\text{IN}}[n] d[nT_s - \tau] e^{-j2\pi f_d n T_s} \ , \qquad (8)$$

$x_{\text{IN}}[n]$ is a complex vector containing I&Q samples of the received signal, T_s is the sampling period, τ is the code phase of the received signal with respect to a local reference, f_d is the Doppler shift, N is the number of samples in a spreading code (4 ms for E1), and $d[n]$ is a locally generated reference. The user can also configure the shape of $d[n]$, allowing simplifications that reduce the computational load. As shown in Figure 2, in narrowband receivers the CBOC waveform can be substituted by a sinBOC modulation with very small performance penalty [6]. For the E1B signal component, the reference signals available in our implementation are:

$$d_{E1B}^{(\text{CBOC})}[n] = \sum_{l=-\infty}^{+\infty} C_{E1B}\left[|l|_{4092} \right] p(t - lT_{c,E1B}) \cdot \cdot (\alpha sc_A[n] + \beta sc_B[n]) \ , \qquad (9)$$

$$d_{E1B}^{(\text{sinBOC})}[n] = \sum_{l=-\infty}^{+\infty} C_{E1B}\left[|l|_{4092} \right] p(t - lT_{c,E1B}) sc_A[n] \ , \qquad (10)$$

while for E1C, users can choose among:

$$d_{E1C}^{(\text{CBOC})}[n] = \sum_{m=-\infty}^{+\infty} \sum_{l=1}^{4092} C_{E1Cp}\left[l \right] \cdot p[n - mT_{c,E1Cs} - lT_{c,E1Cp}] \cdot \cdot (\alpha sc_A[n] + \beta sc_B[n]) \ , \qquad (11)$$

$$d_{E1C}^{(\text{sinBOC})}[n] = \sum_{m=-\infty}^{+\infty} \sum_{l=1}^{4092} C_{E1Cp}\left[l \right] \cdot p[n - mT_{c,E1Cs} - lT_{c,E1Cp}] \cdot \cdot sc_A[n]. \qquad (12)$$

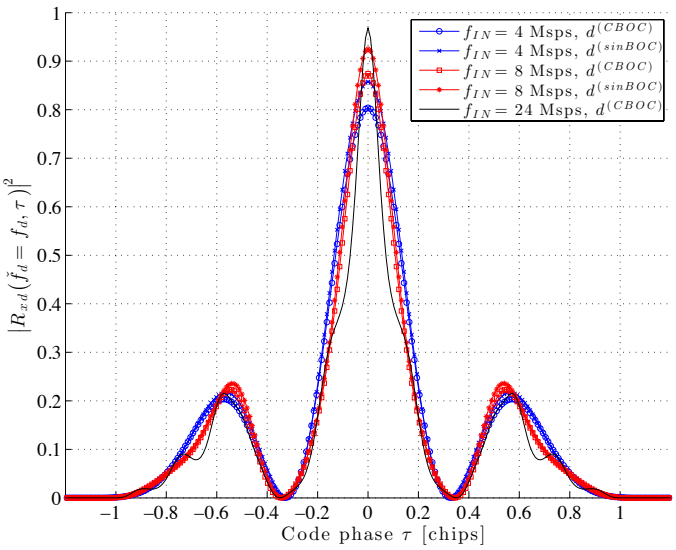

Fig. 2. Normalized $\left|R_{xd}\left(\check{f}_d = f_d, \tau\right)\right|^2$ for different sampling rates and local reference waveforms.

The maximization in (7) requires a two-dimensional search in a function which output results from a multiplication-and-sum of N complex samples, becoming the computational bottleneck of the whole process. A usual method to alleviate this issue is to resort to the FFT-based circular convolution, which exchanges the expensive multiplication-and-sum operation by a discrete Fourier transform, a vector product and an inverse transform, taking advantage of the efficient implementations available for such operations [5].

The magnitude of (8), also known as cross-ambiguity function, is also used to decide whether the satellite corresponding to the local replica $d[n]$ is in view or it is not. Resorting to signal detection theory, it is possible to define tests statistics with desirable properties. A criterion commonly used for a detection problem is the maximization of the detection probability (P_d) subject to a given false alarm probability (P_{fa}). It is well-known in the literature that the optimum solution to that problem can be found by applying the Neyman-Pearson (NP) approach, which requires perfect knowledge of the signal parameters and constitutes the uniformly most powerful test [7]. Assuming additive white Gaussian noise and replacing the true synchronization parameters by their ML estimators in the NP detector, one obtains the Generalized Likelihood Ratio Test (GLRT) function, that can be written as:

$$T_{\text{GLRT}}\left(\mathbf{x}_{\text{IN}}\right) = \max_{f_d,\tau}\left\{\frac{\left|\hat{R}_{xd}(f_d,\tau)\right|^2}{\hat{R}_{xx}}\right\}, \qquad (13)$$

where $\hat{R}_{xx}$ is an estimation of the input signal power. It can be shown [7] that this acquisition test statistic is a Constant False Alarm Rate (CFAR) detector because P_{fa} does not depend on the noise power.

Our implementation is described in Algorithm 1. The computation of the Fast Fourier Transform and its inverse (steps 4 and 6) are performed by means of GNU Radio wrappers of the FFTW3 library [8], an efficient implementation for computing the discrete Fourier transform, and the product in step 5 is implemented with the Vector-Optimized Library of Kernels (VOLK) [9], which generates processor-specific Single-Input Multiple-Data (SIMD) assembly instructions that take advantage of parallel computing techniques and allows writing efficient and portable code.

Algorithm 1 Parallel Code Phase Search (PCPS) acquisition algorithm.

Require: Input signal buffer $\mathbf{x}_{\text{IN}}$ of N complex samples, provided by the Signal Conditioner; on-memory FFT of the local replica, $D[n] = FFT_N\{d[n]\}$; acquisition threshold γ; freq. span $[f_{min}\ f_{max}]$; freq. step f_{step}.

Ensure: Decision positive or negative signal acquisition. In case of positive detection, it provides coarse estimations of code phase $\hat{\tau}_{acq}$ and Doppler shift $\hat{f}_{d_{acq}}$ to the Tracking block.

1: Compute input signal power estimation: $\hat{P}_{in} = \frac{1}{N}\sum_{n=0}^{N-1}|x_{\text{IN}}[n]|^2$.
2: **for** $\check{f}_d = f_{min} : f_{step} : \check{f}_d = f_{max}$ **do**
3: Carrier wipe-off: $x[n] = x_{\text{IN}}[n] \cdot e^{-(j2\pi\check{f}_d n T_s)}$, for $n = 0, ..., N-1$.
4: $X[n] = \text{FFT}_N\{x[n]\}$
5: $Y[n] = X[n] \cdot D[n]$, for $n = 0, ..., N-1$.
6: $R_{xd}(\check{f}_d, \boldsymbol{\tau}) = \frac{1}{N^2}\text{IFFT}_N\{Y[n]\}$
7: **end for**
8: Search maximum and its indices in the search grid: $\{S_{max}, f_i, \tau_j\} \Leftarrow \max_{f,\tau}|R_{xd}(f,\tau)|^2$
9: Compute the GLRT function with normalized variance: $\Gamma_{\text{GLRT}} = \frac{2 \cdot N \cdot S_{max}}{\hat{P}_{in}}$
10: **if** $\Gamma_{\text{GLRT}} > \gamma$ **then**
11: Declare positive acquisition and provide $\hat{f}_{d_{acq}} = f_i$ and $\hat{\tau}_{acq} = \tau_j$.
12: **else**
13: Declare negative acquisition.
14: **end if**

On top of Algorithm 1 (or any other providing the same outputs), we could integrate results from more than one consecutive code periods in order to enhance the *acquisition sensitivity*, understood as the minimum signal power at which a receiver can correctly identify the presence of a particular satellite signal in the incoming RF signal within a given time-out interval.

D. Tracking

The Tracking block is also receiving the data stream x_{IN}, but does nothing until it receives a "positive acquisition" message from the control plane, along with the coarse estimations $\hat{\tau}_{acq}$ and $\hat{f}_{d_{acq}}$. Then, its role is to refine such estimations and track their changes along the time. Three parameters are relevant for signal tracking: the evolution of the code phase τ, Doppler shift f_d and carrier phase ϕ. According the ML principle

expressed in (7), obtaining the optimum estimators implies the maximization of the correlation of the incoming signal with its matched filter. This is usually achieved with closed-loop structures designed to minimize the difference between the code phase, carrier phase and frequency of the incoming signal with respect to a locally-generated replica.

In the case of code phase tracking, the cost function is driven to the maximum using feedback loops that employ the derivative $\frac{dR_{xd}(\tau)}{d\tau}$ zero-crossing as a timing error detector. This is the case of the Delay Lock Loop (DLL) architecture and its wide range of variants usually applied for GPS L1 signals, where the receiver computes three samples of R_{xd}, usually referred to as *Early* $E = R_{xd}(\hat{\tau} - \epsilon)$, *Prompt* $P = R_{xd}(\hat{\tau})$ and *Late* $L = R_{xd}(\hat{\tau} - \epsilon)$, with ϵ ranging from $0.1T_c$ to $0.5T_c$, and then computes a timing error with some combination of those samples, known as discriminator functions. In case of Galileo E1, the CBOC(6,1,$\frac{1}{11}$) modulation creates correlation ambiguities, as shown in Figure 2. The possibility of tracking a local maximum instead of the global one can be avoided by using discriminators that consider two extra samples of the cost function, referred to as *Very Early* $\mathrm{VE} = R_{xd}(\hat{\tau} - \epsilon')$ and *Very Late* $\mathrm{VL} = R_{xd}(\hat{\tau} + \epsilon')$, with $\epsilon' > \epsilon$.

In the case of carrier tracking loops, the pilot channel E1C can be used for the phase error estimation, since it does not contain data bit transitions, and, theoretically, coherent integration of several code periods can be used for as long as needed. As a consequence, a discriminator that is insensitive to phase jumps can be used. Using pure PLL tracking on the pilot channel as well as longer coherent integration improves the carrier *tracking sensitivity*, the minimum signal power at which the receiver can keep the tracking process in lock.

In addition to track the synchronization parameters, the Tracking block must also implement code and carrier lock detectors, providing indicators of the tracking performance.

Our implementation is described in Algorithm 2. The computation of the complex values VE, E, P, L and VL in step 4 was implemented using the VOLK library. The PLL discriminator implemented in step 5 is the extended arctangent (four-quadrant) discriminator, and for the DLL we used the normalized Very Early Minus Late Power discriminator proposed in [10] (step 9). For code lock detection (step 12), we used the Squared Signal-to-Noise Variance (SNV) estimator proposed in [11]. In the case of carrier lock detection (step 13), we used the normalized estimate of the cosine of twice the carrier phase [12]. The values of the lock indicator range from -1, when the locally generated carrier is completely out of phase, to 1, that indicates a perfect match. When either the code or the carrier detectors are below given thresholds during a consecutive number of code periods ϑ, the Tracking block informs to control plane through the message queue.

E. Decoding of the navigation message

As shown in Equation (5), the E1B signal component carries the I/NAV navigation message, which provides the space vehicle identification (SVID), an Issue of Data, the ephemeris data, a signal-in-space accuracy (SISA) indicator, clock correction parameters, a ionospheric correction, the Broadcast Group Delay (BGD), signal health and data validity status, Galileo System Time (GST), GST-UTC and GST-GPS time conversion parameters, the almanacs, and Search and Rescue (SAR) service data. The message structure is depicted in Figure 3: each frame contains 24 subframes, and each subframe contains 15 nominal pages, having a duration of 2 seconds transmitted sequentially in time in two parts of duration 1 second each. Each page part (denoted as *even* or *odd*) consists of a 10-bit synchronization pattern followed by 240 coded symbols, corresponding to 114 data bits and 6 tail bits (sequence of zeros) that allow Viterbi decoding. Three levels of error coding are applied to the Galileo message data stream: *i)* a Cyclic Redundancy Check (CRC) with error detection capabilities after recovery of the received data, *ii)* a one-half rate Forward Error Correction (FEC) convolutional coding, and *iii)* block interleaving on the resulting frames, with 30 columns (where data is written) and 8 rows (where data is read), providing robustness to the FEC decoding algorithm by avoiding error bursts.

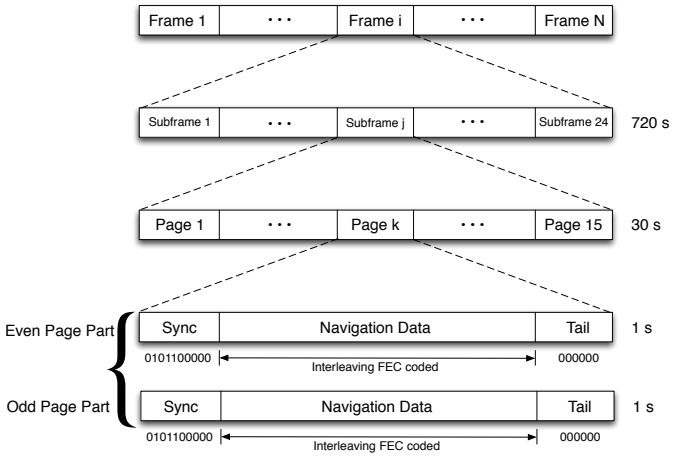

Fig. 3. I/NAV navigation message structure.

F. Connection to third-party software

Each processing block can be configured to dump internal data to a file, thus allowing the use of tools such as Octave or Matlab for analysis, graphical visualization and debugging purposes. Provided that GNSS-SDR is designed to easily accommodate new algorithms, we cannot ignore the convenience of other third-party tools for rapid prototyping widely used by the industry, such as Mathworks' Matlab/Simulink or National Instruments' LabView. These are commercial tools, often with machine-related licenses that constrain the use of the software to certain computers, maybe remote to the user's machine. In order to use them for algorithm validation within the framework of GNSS-SDR, we developed a so-called Transmission Control Protocol (TCP) connector, that allows by-passing internal processing blocks by external ones, run by other software tools and, optionally, in a remote computer.

This TCP connector allows, for instance, to perform Simulink model-in-the-loop testing, using signal samples from a file and integrating the model in a complete GNSS receiver. In such a setup, the developer enjoys the design environment, block libraries and tools for modeling, simulating and analyzing dynamic systems provided by Simulink, while using GNSS-SDR for rapidly integrating the Simulink model into a full receiver, which will make use of that model just as for any other of the existing "native" loops. However, the TCP connection severely slows down the execution of the whole receiver, preventing its use in real-time mode. In practice, it can be used with GNSS-SDR in post-processing mode only, that is, capturing raw bits at the RF front-end's ADC output, storing them in a file, and then processing the signals offline.

IV. Experimental Results with Real–Life Signals

The software receiver described so far was tested using real signals broadcast by the in-orbit Galileo satellites. The hardware front-end consisted of an active antenna (Novatel GPS-600), which has a 3 dB pass at E1 of 1575 ± 8 MHz and a built-in low noise amplifier that provides 28 ± 3 dB of gain. The antenna was connected to a DBSRX daughterboard [13], in charge of amplification, downconversion and filtering, followed by a USRP v1 Rev. 4.5 [14] for analog-to-digital conversion, fine downconversion and decimation, and USB 2.0 interface.

The DBSRX is a complete receiver system for 800 MHz to 2.3 GHz with a 3-5 dB noise figure. The DBSRX features a software controllable channel filter which can be made as narrow as 1 MHz, or as wide as 60 MHz. After the antenna SMA input there is a 17 dB amplifier, followed by a programmable (0-56 dB) amplifier, feeding two (I&Q) direct conversion tuners (Maxim's MAX2118, covering from 925 to 2175 MHz) followed by a another programmable (0-24 dB) amplifier, a programmable lowpass filter (2 to 33 MHz) and a switchable 5 dB amplifier. The output signals are connected to the USRP, that ships with a voltage controlled, temperature compensated crystal oscillator as a 64 MHz reference clock ($\pm$20 ppm) and samples the I&Q signal components with an Analog Devices' AD9862 (14 bits per sample, $f_s = 64$ Msps). Then, an Altera Cyclone EP1C12Q240C8 FPGA performs digital downconvertion to baseband with programmable decimation rates, and finally sends the digital signal through USB 2.0 (which provides a 32 MBps bandwidth) to the computer running the software receiver.

The host computer was an Intel Core2 Quad Q9400 processor at 2.66 GHz with 4 GB of RAM, running Linux Ubuntu 12.04 32 bits as operating system and GNU Radio 3.6.2. The Signal Source block was configured with the UHD, setting $f_c = 1575,420$ MHz and an amplification of 60 dB (the driver internally adjusts the programmable amplifiers in order to get a low the noise figure while avoiding ADC saturation), using decimation rates of 8 and 16 (corresponding to $f_{IN} = 8$ Msps and $f_{IN} = 4$ Msps), and then capturing samples in a hard disk using a capture-to-file utility of the UHD driver.

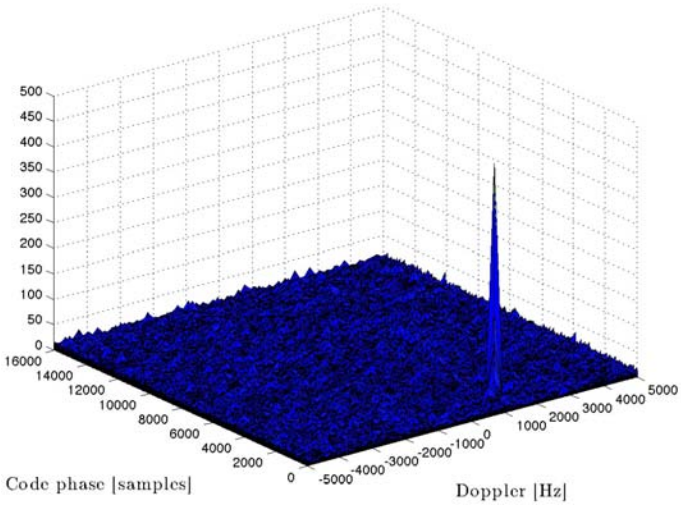

Fig. 4. GLRT statistic of Parallel Code Phase Search acquisition algorithm for a configuration of $f_{\text{IN}} = 4$ Msps, a frequency span of $\pm$5 kHz with steps of 250 Hz, and using the E1B sinBOC local replica for Galileo's IOV satellite PRN 11.

Then, we configured the Signal Source to read from the corresponding file, thus avoiding real-time constraints and delivering a stream of complex, baseband samples with real and imaginary parts in `float` data type (IEEE-754 32 bit floating point). The Signal Conditioner was deactivated (*pass–through*) and the Channels were configured to use Algorithm 1 for Acquisition and Algorithm 2 for Tracking.

The parameters for acquisition were set as $T_{int} = 4$ ms, $N = T_{int} f_{\text{IN}}$ (thus $N = 32000$ and $N = 16000$ samples per code period, corresponding to a receiver bandwidth of 4 MHz and 8 MHz, respectively), $\gamma = 50$, $f_{d_{min}} = -5$ kHz, $f_{d_{max}} = 5$ kHz and $f_{step} = 250$ Hz. The local reference waveform were $d_{E1B}^{(\text{sinBOC})}[n]$, defined in (10), for the Galileo satellite identified with PRN 11. Figure 4 shows the search grid computed in steps 2-7 of Algorithm 1. Similar results were obtained when using $d_{E1B}^{(\text{CBOC})}[n]$, $d_{E1C}^{(\text{sinBOC})}[n]$ or $d_{E1C}^{(\text{CBOC})}[n]$, for both bandwidth configurations.

The tracking algorithm was configured with $T_{int} = 4$ ms, $\mathcal{T} = 0.85$, $\epsilon = 0.15 \cdot T_{c,E1B}$, $\epsilon' = 0.6 \cdot T_{c,E1B}$, $\mathcal{U} = 10$, $CN0_{min} = 25$ dB-Hz, $\vartheta = 20$, $\text{BW}_{DLL} = 15$ Hz, and $\text{BW}_{PLL} = 2$ Hz (see description of those parameters in Algorithm 2). Figure 5 shows the evolution in time of the correlators VE, E, P, L and VL obtained with such configuration.

In order to provide a simple example of the possibilities of the TCP connector described in Section III-F, we configured the software receiver to use a tracking block implemented in Simulink instead of a C++ implementation, with the same parameters. We established a TCP connection just after the computation of the correlation samples VE, E, P, L, and VL (step 4 in Algorithm 2), and the DLL/PLL discriminators and filters (steps 5–11 in Algorithm 2) were implemented in a Simulink model, as shown in Figure 6. Some details about the implementation of the block labeled as *Galileo E1 Tracking*

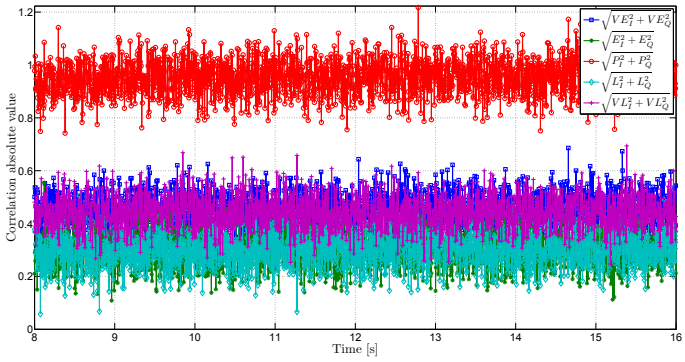

Fig. 5. Evolution of VE, E, P, L and VL computed with a C++ implementation of the Tracking block.

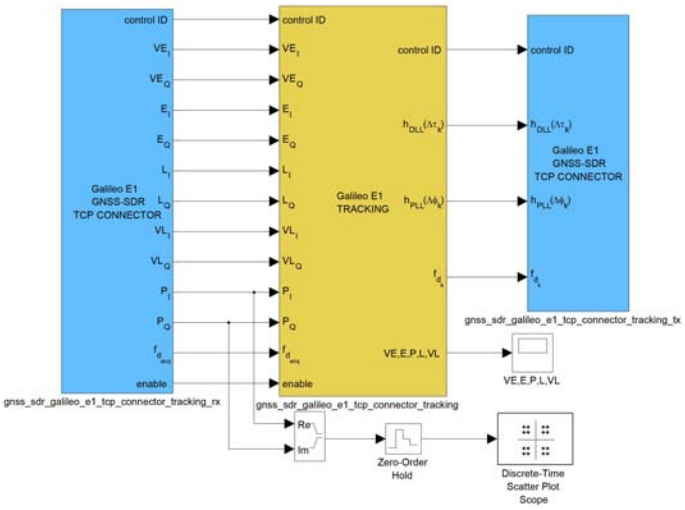

Fig. 6. Simulink model-in-the-loop testing using GNSS-SDR and the TCP connectors.

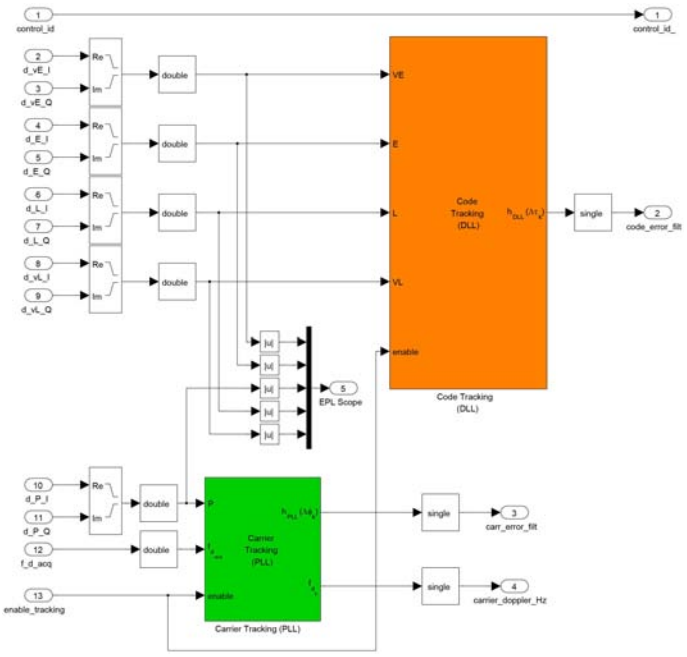

Fig. 7. Implementation details of the Simulink model of a Galileo E1 Tracking block (the yellow block in Figure 6).

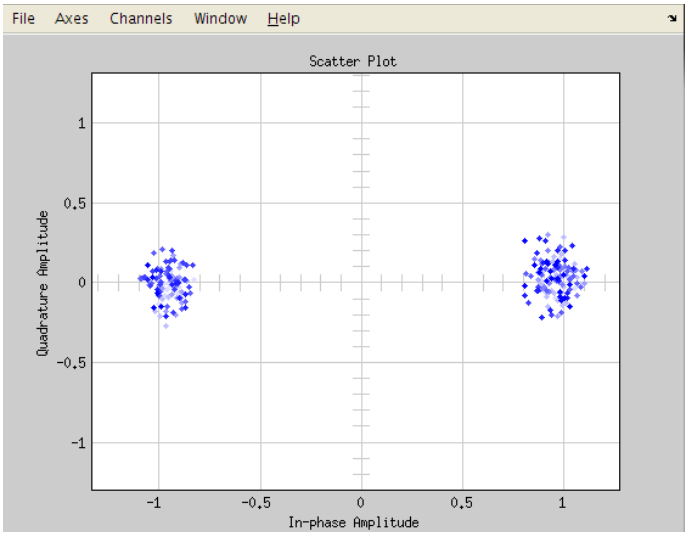

Fig. 8. Screen capture of the discrete-time scatter plot scope of Figure 6.

are shown in Figure 7. A TCP connector returned back the results of the Simulink block to the software receiver, which continued with the processing flow in the same way it does for any other block implementation. Figure 8 shows the obtained scatter plot, showing the data bits of signal E1B.

Finally, the obtained navigation bits are decoded by applying a de-interleaver, a Viterbi decoder and a CRC, as explained in Section III-E. Figure 9 shows the correlation of the data bits and the preamble (10-bit synchronization pattern), showing the expected periodicity of 1 s (corresponding to the start of each data page, as shown in Figure 3).

V. Conclusions

In this paper, we presented an open source software radio implementation in C++ of the acquisition, tracking and navigation message decoding of Galileo E1B and E1C signals. We discussed the main signal processing aspects of such stages, and experimental results using signals coming from the in-orbit Galileo satellites validated the approach. The receiver software architecture was designed to allow several implementations for each processing block, as long as they follow a minimal common interface. As an example of such flexibility, we implemented a tracking algorithm in Simulink and performed model-in-the-loop testing using TCP connectors for the communication between the software receiver and an external third-party application. The documentation and the source code of the software receiver described in this work, as well as ongoing developments, are freely available at http://gnss-sdr.org.

Acknowledgment

This work has been partially funded by the Google Summer of Code 2012 program (GSoC 2012).

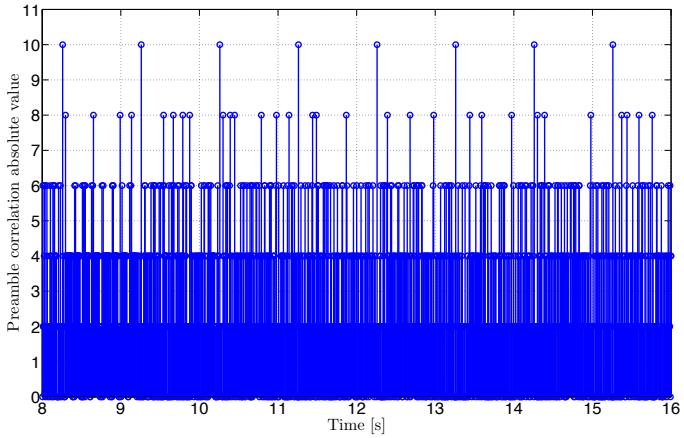

Fig. 9. Correlation of the data bits and the 10-bit synchronization pattern. The peaks spaced 1 s apart mark the starting of the data pages.

REFERENCES

[1] C. Fernández–Prades, C. Avilés, L. Esteve, J. Arribas, and P. Closas, "Design patterns for GNSS software receivers," in *Proc. of the 5th ESA Workshop on Satellite Navigation Technologies (NAVITEC'2010)*, ESTEC, Noordwijk, The Netherlands, Dec. 2010, DOI:10.1109/NAVITEC.2010.5707981.

[2] C. Fernández–Prades, J. Arribas, P. Closas, C. Avilés, and L. Esteve, "GNSS-SDR: An open source tool for researchers and developers," in *Proc. of the ION GNSS 2011 Conference*, Portland, Oregon, Sept. 2011.

[3] "GNU Radio," http://gnuradio.org/redmine/wiki/gnuradio, Retrieved: July 31, 2012.

[4] European Union, *European GNSS (Galileo) Open Service. Signal In Space Interface Control Document. Ref: OS SIS ICD, Issue 1.1*, September 2010.

[5] K. Borre, D. M. Akos, N. Bertelsen, P. Rinder, and S. H. Jensen, *A Software–Defined GPS and Galileo Receiver. A Single–Frequency Approach*, Applied and Numerical Harmonic Analysis. Birkhäuser, Boston, MA, 2007.

[6] E. Simona Lohan, "Limited bandwidths and correlation ambiguities: Do they co-exist in Galileo receivers," *Positioning*, vol. 2, no. 1, pp. 14–21, Feb. 2011, DOI: 10.4236/pos.2011.21002.

[7] S. M. Kay, *Fundamentals of statistical signal processing, Volume II: Detection theory*, Prentice Hall, Upper Saddle River, NJ, 1998.

[8] M. Frigo and S. G. Johnson, "The design and implementation of FFTW3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005, Special issue on "Program Generation, Optimization, and Platform Adaptation".

[9] "VOLK," http://gnuradio.org/redmine/projects/gnuradio/wiki/Volk, Retrieved: October 14, 2012.

[10] A. Jovanovic, C. Mongrédien, Y. Tawk, C. Botteron, and P. A. Farine, "Two-Step Galileo E1 CBOC Tracking Algorithm: When Reliability and Robustness Are Keys!," *International Journal of Navigation and Observation*, 2012, Article ID 135401. Special issue on "Advances in Signal Tracking for GNSS Receivers: Theory and Implementation". DOI:10.1155/2012/135401.

[11] D. R. Pauluzzi and N. C. Beaulieu, "A comparison of SNR estimation techniques for the AWGN channel," *IEEE Transactions on Communications*, vol. 48, no. 10, pp. 1681–1691, Oct. 2000.

[12] A. J. Van Dierendonck, "GPS Receivers," in *Global Positioning System: Theory and Applications*, B. W . Parkinson and J. J. Spilker, Jr., Eds., vol. I, chapter 8, pp. 329–407. American Institute of Aeronautics and Astronautics, Inc., Washington, DC, 1995.

[13] Ettus Research, "Product Detail: DBSRX2 800-2300 MHz Rx," http://www.ettus.com/product/details/DBSRX2, Retrieved: October 24, 2012.

[14] Ettus Research, "USRP1," http://www.ettus.com/product/details/USRP-PKG, Retrieved: October 24, 2012.

Algorithm 2 Implemented tracking algorithm.

Require: Complex sample stream, $\mathbf{x}_{\text{IN}}$; estimations of code phase $\hat{\tau}_{acq}$ and Doppler shift $\hat{f}_{d_{acq}}$; buffer size for power estimation, $\mathcal{U}$; carrier lock detector threshold, $\mathcal{T}$; $CN0_{min}$; maximum value for the lock fail counter, ϑ; correlators spacing ϵ and ϵ'; loop filters bandwidth BW_{DLL} and BW_{PLL}; integration time T_{int}.

Ensure: Track signal's synchronization parameters within a given lock margin. Inform about a loss of lock.

Initialization: Using $\hat{\tau}_{acq}$ and a sample counter $\mathcal{N}$, skip samples until $\mathbf{x}_{\text{IN}}$ is aligned with local PRN replica. Set $\upsilon = 0$, $k = 0$, $\hat{f}_{d_0} = \hat{f}_{d_{acq}}$, $\hat{\phi}_0 = 0$, $\psi_1 = 0$, $N_1 = \text{round}(T_{int} f_{\text{IN}})$.

Increase the integration period counter: $k = k + 1$.

Generate local code references: for $n = 1...N_k$, $s[n] = d_{E1B/E1C_p}\left[\text{round}(\delta_k \cdot n + \psi_k)\right]$, where $\delta_k = \frac{1}{T_{c,E1B} \cdot f_{\text{IN}}} \left(1 + \frac{\hat{f}_{d_{k-1}}}{f_c^{(Gal\ E1)}}\right)$, and the Very Early, Early, Late, and Very Late versions with ϵ and ϵ'.

Generate local carrier: for $n = 1...N_k$, $c[n] = e^{-j\left(2\pi \hat{f}_{d_{k-1}} \frac{n}{f_{\text{IN}}} + \text{mod}\left(\hat{\phi}_{k-1}, 2\pi\right)\right)}$.

Perform carrier wipe-off and compute the complex samples VE_k, E_k, P_k, L_k and VL_k. Example: $\text{P}_k = \frac{1}{N_k} \sum_{n=0}^{N_k - 1} x_{\text{IN}}[n] s[n] c[n]$.

Compute PLL discriminator: $\Delta\hat{\phi}_k = \text{atan2}\left(\frac{\text{P}_{Q_k}}{\text{P}_{I_k}}\right)$.

Filter $\Delta\hat{\phi}_k$ with a bandwidth BW_{PLL}: $h_{PLL}\left(\Delta\hat{\phi}_k\right)$.

Update carrier frequency estimation (in Hz): $\hat{f}_{d_k} = \hat{f}_{d_{acq}} + \frac{1}{2\pi T_{int}} h_{PLL}\left(\Delta\hat{\phi}_k\right)$.

Update carrier phase estimation (in rad): $\hat{\phi}_k = \hat{\phi}_{k-1} + 2\pi \hat{f}_{d_k} T_{int} + h_{PLL}(\Delta\hat{\phi})$.

Compute DLL discriminator: $\Delta\hat{\tau}_k = \frac{\mathcal{E}_k - \mathcal{L}_k}{\mathcal{E}_k + \mathcal{L}_k}$, where: $\mathcal{E}_k = \sqrt{\text{VE}_{I_k}^2 + \text{VE}_{Q_k}^2 + E_{I_k}^2 + E_{Q_k}^2}$, and $\mathcal{L}_k = \sqrt{\text{VL}_{I_k}^2 + \text{VL}_{Q_k}^2 + L_{I_k}^2 + L_{Q_k}^2}$.

Filter $\Delta\hat{\tau}_k$ with a bandwidth BW_{DLL}: $h_{DLL}\left(\Delta\hat{\tau}_k\right)$.

Update code phase estimation (in samples): $N_{k+1} = \text{round}(S)$ and $\psi_{k+1} = S - N_{k+1}$, where $S = \frac{T_{int} f_{\text{IN}}}{\left(1 + \frac{\hat{f}_{d_k}}{f_c^{(Gal\ E1)}}\right)} + \psi_k + h_{DLL}(\hat{\Delta\tau}_k) f_{\text{IN}}$.

Code lock indicator: $\hat{\text{C/N0}} = 10 \cdot \log_{10}(\hat{\rho}) + 10 \cdot \log_{10}(\frac{f_{IN}}{2}) - 10 \cdot \log_{10}(L_{\text{PRN}})$, where: $\hat{\rho} = \frac{\hat{P}_s}{\hat{P}_n} = \frac{\hat{P}_s}{\hat{P}_{tot} - \hat{P}_s}$, $\hat{P}_s = \left(\frac{1}{\mathcal{U}} \sum_{i=0}^{\mathcal{U}-1} |\text{P}_{I_{k-i}}|\right)^2$, and $\hat{P}_{tot} = \frac{1}{\mathcal{U}} \sum_{i=0}^{\mathcal{U}-1} |\text{P}_{k-i}|^2$.

Phase lock indicator: $T_{carrier} = \frac{\left(\sum_{i=0}^{\mathcal{U}-1} \text{P}_{I_{k-i}}\right)^2 - \left(\sum_{i=0}^{\mathcal{U}-1} \text{P}_{Q_{k-i}}\right)^2}{\left(\sum_{i=0}^{\mathcal{U}-1} \text{P}_{I_{k-i}}\right)^2 + \left(\sum_{i=0}^{\mathcal{U}-1} \text{P}_{Q_{k-i}}\right)^2}$.

if $T_{carrier} < \mathcal{T}$ or $CN0 < CN0_{min}$ **then**

  Increase lock fail counter $\upsilon \leftarrow \upsilon + 1$.

else

  Decrease lock fail counter $\upsilon \leftarrow \max(\upsilon - 1, 0)$.

end if

if $\upsilon > \vartheta$ **then**

  Notify the loss of lock to the control plane through the message queue.

end if

Output: P_k, accumulated carrier phase error $\hat{\phi}_k$, code phase $\mathcal{N} \leftarrow \mathcal{N} + N_k + \psi_k$, $\hat{\text{C/N0}}$.