REAL-TIME SOFTWARE DEFINED GPS RECEIVER

A Bachelor Thesis

Submitted to the Faculty

of

Communication and Environment

of

Hochschule Rhein-Waal

by

German Abdurahmnaov

November 2021

Friedrich-Heinrich-Allee 33

REAL-TIME SOFTWARE DEFINED GPS RECEIVER

A Thesis Submitted in

Partial Fulfillment of the

Requirements of the Degree of

Bachelor of Science

in

Communication and Information Engineering

Submitted to the Faculty

of

Communication and Environment

of

Hochschule Rhein-Waal

by

German Abdurahmnaov

Under Supervision

of

Prof. Dr. Volker Strumpen

Friedrich-Heinrich-Allee 33

Matriculation Number:

24500

Submission Date:

22.11.2021

ACKNOWLDEGMENTS

My supervisor Prof. Dr. Volker Strumpen has contributed significantly to this work. He took the time to meet with me nearly every week to check on the status and all the technical issues I was facing. We discussed ideas, possible solution for problems and much more.
I would also like to thank the people from amazing Gnu Radio and GNSS-SDR communities, who provided immense support from the very beginning to the very end.

And I supposed I need to include my brother, Roman Abdurahmanov, without whom pursuing this bachelor's degree would be impossible. I have also used to bounce my ideas of off him quite a lot.

TABLE OF CONTENTS

ABSTRACT

The Global Positioning System (GPS) is an open access satellite navigation system widely integrated into daily life. Designing a GPS receiver is a challenging process and commonly requires specific hardware design for different operating modes and environments. The objective of the Software Defined GPS receiver is to achieve one reconfigurable system which can adapt to various application requirements.

The first software implementation of a GPS receiver was described by (Akos, 1997). It was then compiled in a book by (Borre, et al., 2007) along with a non-real time MATLAB-based GPS Receiver implementation. The motivation for this work is the expansion of this non-real time GPS. The new receiver should be structured to process streaming real time data coming from a RF front end or a GPS signal recording file.

The receiver for this thesis was developed as an external module for Gnu Radio (Wiki GNU Radio, 2021). Gnu radio is a constantly growing and well-maintained open-source project, that provides an ability to create custom signal processing blocks, flowgraph GUI to build signal processing systems and great debugging and data visualization tools. It is widely used in number of academia, government, and hobbyist environments, has a well-developed community and thus is a perfect fit for the purpose.

At the time of writing, receiver is capable of processing legacy civil L1 GPS signal, from a real time data stream simulated in Gnu Radio from a recorded GPS signal file.

The source files for the Gnu Radio module can be found at:
https://github.com/geraman21/gnss_grc

1 SOFTWARE DEFINED GPS RECEIVER OVERVIEW

1.1 GPS Signal

The Global Positioning System consists of 32 satellites, each orbiting earth approximately once every 12 hours. The orbits were designed such that at every given time at least 6 satellites are observable from earth. This chapter provides a basic overview of the GPS signal that each of the satellites is transmitting.

The GPS signals are transmitted on two frequencies in Ultra High Frequency (UHF) band, L1 and L2. This thesis focuses on the civilian-access signal, known as Coarse Acquisition (C/A) code which is broadcasted on L1 GPS carrier at 1.57542 GHz. It is derived from a common frequency $f_0 = 10.23MHz$ (Borre, et al., 2007, p. 17):

$$f_{L1} = 154 f_0 = 1575.42MHz$$

The signal is composed of three parts:

Carrier wave, with frequency f_{L1}, Navigation data and spreading sequence.

1.1.1 Navigation Data

The navigational message broadcasted from any GPS satellite is a 50Hz signal with a fixed predefined structure. This structure is described below.

Navigation data is characterized by a 50bps data sequence which is uploaded to all satellites from ground stations in the GPS Control Segment. It is formatted into 30-bit words. Words are grouped into subframes. Every subframe consists of 10 words, thus, is 300 bits in length and exactly 6s in duration. 5 subframes construct a complete navigation data frame, which totals 1500 bits and is 30s in duration. Satellite's orbital parameters are contained withing first three subframes of a frame. Subframe 4 and 5 contains an atmospheric correction model, and coarse satellite positioning parameters known as almanac (Seung-Hyun, et al., 2008).

1.1.2 Spreading sequence

To identify each individual satellite in the received GPS signal, navigation data is combined with a predefined spreading sequence unique for each satellite. This section provides a brief overview of these sequences.

Each satellite generates two unique Pseudo Random Noise (PRN) codes. The first one is the C/A code. The second one is called encrypted precision code (P(Y)). To process the

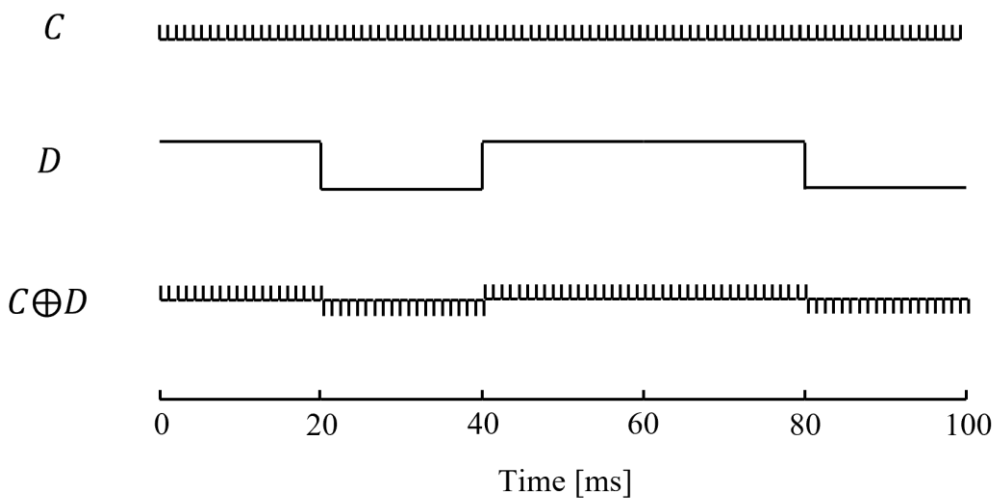

Figure 1. Combination of the C/A code with the navigation data for one satellite. Here C is the C/A Code, D is the navigation message, and $C \oplus D$ is the combined signal using exclusive or binary operator. Every millisecond of the signal contains one full C/A sequence.

L1 signal it is enough to understand the structure of the C/A code. Each C/A code is a unique 1023 long sequence measured in chips rather than bits. While individual chip carries no information the C/A code in its entirety carries one navigation bit. The chip rate of the C/A code is 1.023MHz. Therefore, the C/A code repeats every millisecond. If +1 navigation bit must be transmitted C/A codes are transmitted. If -1 bit must be transmitted, inverted C/A codes are transmitted (Figure 1).

Each C/A code is taken from a family of 37 codes known as Gold codes. First 32 codes are assigned to 24 active satellites and are recycled, when old satellites die, and new ones are launched. Uniqueness of these codes makes it possible to identify each satellite from the constellation (Borre, et al., 2007, p. 18).

1.1.3 Signal structure and modulation

After C/A code and navigation message are combined, signal is modulated onto the carrier signal using binary phase shift keying (BPSK) method.

The signal transmitted from a satellite k can be described as:

$$\begin{aligned} s_k(t) = {} & \sqrt{2P_c}\big(C_k(t) \oplus D_k(t)\big)\cos(2\pi f_{L1} t) \\ & + \sqrt{2P_{PL1}}\big(P_k(t) \oplus D_k(t)\big)\sin(2\pi f_{L1} t) \\ & + \sqrt{2P_{PL2}}\big(P_k(t) \oplus D_k(t)\big)\sin(2\pi f_{L2} t) \end{aligned} \qquad \textit{Eq. 1}$$

Where $\oplus$ is an exclusive OR operation, P_c, P_{PL1}, and P_{PL2} are the powers of signals with C/A or P code, C_k is the C/A code sequence assigned to satellite number k, P_k is the P(Y) code sequence assigned to satellite number k, D_k is the navigation data sequence, and f_{L1} and f_{L2} are the carrier frequencies of L1 and L2, respectively (Borre, et al., 2007, p. 19).

1.2 Receiver Operation

The first step in receiver's operation is to identify all observable satellites. At least 4 satellites are required to compute receiver's position and altitude. Each satellite identified from the signal is assigned to an individual channel. This channel performs all necessary signal processing steps until navigation bits are extracted. Navigation data from each channel is then used in conjunction to compute receiver's position. This section describes the steps necessary to carry through the process described above.

1.2.1 Acquisition

To allocate a satellite to a channel, receiver must first know which satellites are currently identifiable from the signal, i.e., observable in the sky. The process of identifying observable satellites is called acquisition and is described below.

Two most common ways to find the initial set of observable satellites are called warm start and cold start.

Warm start: almanac data and last position computed by the receiver can be used to identify approximate position of all GPS satellites orbiting the earth. At least two major conditions must be met for a successful warm start: receiver should have already calculated position at least once, receiver is within a reasonable range of the last calculated position (Borre, et al., 2007, p. 69).

Cold start: receiver is searching for observable satellites without relying on prerecorded data. The method of searching is commonly referred to as Acquisition (Borre, et al., 2007, p. 69).

Goal of acquisition is to find all visible satellites, their signal strength and coarse values of code phase and carrier frequency. Chapter 2 describes the acquisition process and its implementation in the project.

1.2.2 Carrier and Code Tracking

Main task of signal tracking is to refine coarse values of code phase and carrier frequency provided by acquisition and keep track of them as signal properties change over time (Borre, et al., 2007, p. 87).

Tracking can be split into two parts:

Code tracking is implemented using a delay locked loop (DLL) where total of three local copies of the C/A code, shifted by a predefined number of chips, are created. They are usually referred to as early, prompt, and late versions of C/A code. Correlating these three versions of the code with the incoming signal, identifies the phase of the C/A code and effectively tracks the phase changes in the received signal.

Carrier frequency/phase tracking is implemented using phase locked loop (PLL), which in essence is a feedback loop which compares phases of the reference signal with the phase of the adjustable incoming signal. Once adjustments are made, reference phase is updated, and the process is repeated.

Carrier wave and Code tracking is a continuous process that operates on the raw received signal. This puts great demand on the performance optimization. Carrier and code tracking implementation is discussed in greater details in chapter 3.

1.2.3 Navigation-data extraction

Once channel tracking is established, it is possible to remove the C/A code and carrier wave from the signal, generating navigation data bits on the output.

Once a full navigation frame is received it is possible to extract ephemerides and almanac data of the satellite. Once that is done for at least 4 satellites, receiver can proceed to position computation.

1.2.4 Computation of position

The final objective of a GPS receiver is to compute its position. Knowing time of transmission and time of arrival of a common subframe from at least 4 satellites as well as their ephemeris and almanac parameters allows for position computation. The process of calculating receiver's position is further discussed in Chapter 4.

1.3 GNU Radio

GNU Radio is an open-source software development toolkit that provides number of highly optimized signal processing blocks as well as an ability to create custom blocks, to implement various signal-processing systems. It can be used with external Radio

Frontend (RF) hardware, or without hardware in a simulation-like environment. This section introduces building blocks for Gnu Radio signal processing systems.

1.3.1 Signal Processing Blocks

Gnu radio signal processing blocks are created and further customized using the four general block templates described below.

Blocks in gnu radio are mainly distinguished by the ratio of input to output data rate. When creating a new block there are several types to choose from:

- Synchronous Blocks (1:1)
- Decimation Blocks (N:1)
- Interpolation Blocks (1:M)
- Basic (a.k.a. General) Blocks (N:M)

Where the ratio in parentheses represent the factor of input to output data rate. Thus, a Synchronous block is expected to output the same number of samples it received, Decimator block reduces the output rate by a factor of N and Interpolation block increases the input rate by a factor of M. All the above blocks are derived from the Basic block, which can be thus, fully customized to the user's needs.

1.3.2 Data Flow and Ports

Data flow in gnu radio comes in two different types and is passed between the blocks using corresponding input and output ports.

These types are tagged streams and messages.

Stream: a stream represents a continuous data flow, originated from a source block. A block receiving the stream defines what the output will be if any. Each sample in the stream can be assigned a tag, containing a polymorphic type data (PMT) (Wiki GNU Radio, 2021). Stream is passed to and from a block by the means of stream ports. Each block can have a single, none or many input and output ports. Ports are visually connected with solid arrow lines. Port color represents the data type of the stream such as short, int, float, char, complex etc.

Message: messages in gnu radio are implemented using queuing protocol. Each block has a set of message queues to hold incoming messages and can post messages to message queues of the other blocks. Subscribing is done by the means of message ports

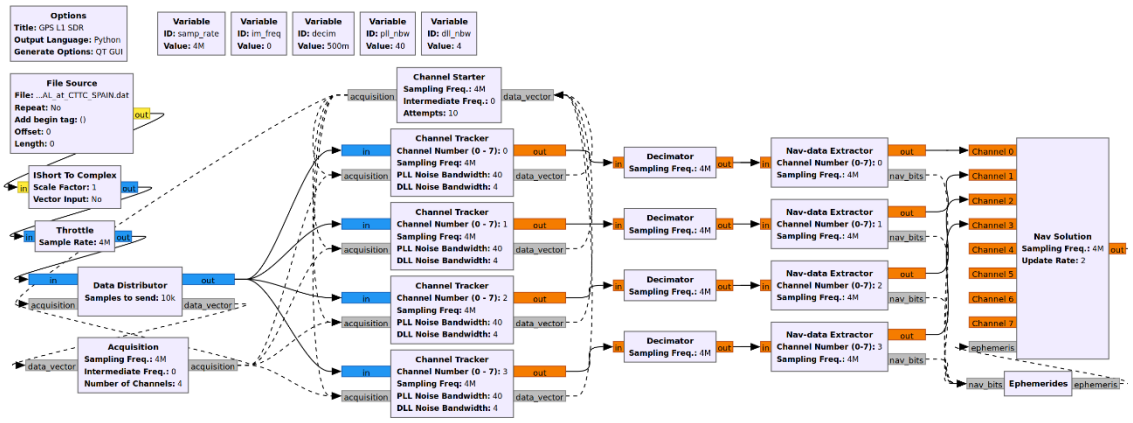

Figure 2. GPS L1 SDR flowgraph built in Gnu Radio Companion, used to process navigation data from 4 satellites and compute receiver's position. Signal source is a 16-bit interleaving complex GPS L1 signal recording sampled at 4MHz.

(Wiki GNU Radio, 2021). All message ports have grey color and generate a dotted arrow lines when connected.

1.3.3 Flowgraph and Gnu Radio Companion (GRC)

GRC is a powerful graphical tool that simplifies the use of Gnu Radio, by allowing users to create Python flowgraphs graphically instead of in code alone (Wiki GNU Radio, 2018). GRC provides a terminal, access to block library and a workspace where these blocks can be connected to construct a functioning signal processing system.

Figure 2 depicts a GPS L1 signal processing flowgraph for positioning in a simulation environment. The "File Source" block points to a binary file containing 16-bit interleaving complex GPS L1 signal sampled at 4MHz. All blocks used in this flowgraph are described in detail in the upcoming chapters.

2 ACQUISITION

As mentioned in section 1.2.1 purpose of acquisition is to determine visible satellites and their coarse values for carrier frequency and code phase. This chapter briefly describes theory behind the acquisition process and dives into its implementation specifics in the project.

2.1 Theoretical background

GPS signal received on the receiver's front end is a combination of signals transmitted from series of different satellites. This section briefly describes the challenges in identifying these satellites and the ways to tackle them.

Signal ***s*** received on the antenna is a combination of signals from all ***n*** visible satellites and can be denoted as:

$$s(t) = s_1(t) + \ s_2(t) + \cdots + \ s_n(t) \qquad \textit{Eq. 2}$$

Where signal $s_n(t)$ can be described by Eq. 1. Thus, to acquire signal s_k from satellite k, signals from all other satellites must be removed. This is achieved by multiplying s_k with a locally generated C/A code corresponding to satellite *k*. C/A codes demonstrate good correlation only for zero lag, meaning that the code can only be properly removed if generated C/A code is perfectly aligned in time with the incoming signal. This time alignment is achieved by finding the correct code phase of the signal. Knowing the length of the C/A code, we compute 1023 possible variations for the code phase (Borre, et al., 2007, p. 70).

After the C/A code has been removed, signal is demodulated by mixing it with locally generated carrier wave. This is done to remove the carrier wave from the incoming signal. To properly remove carrier wave from the signal, frequency of the locally generated carrier wave must be close to the real carrier frequency of the signal. Relative motion of the satellites and the receiver cause Doppler Frequency shift, which can reach as high as ±10kHz from the nominal frequency (Tsui, 2000). To understand whether the satellite is visible it is enough to search the frequency in steps of 500Hz, which results in 41 different frequencies to test (Borre, et al., 2007, p. 71).

Searching through 1023 * 41 possible combinations allows us to identify all visible satellites above a predefined signal strength threshold.

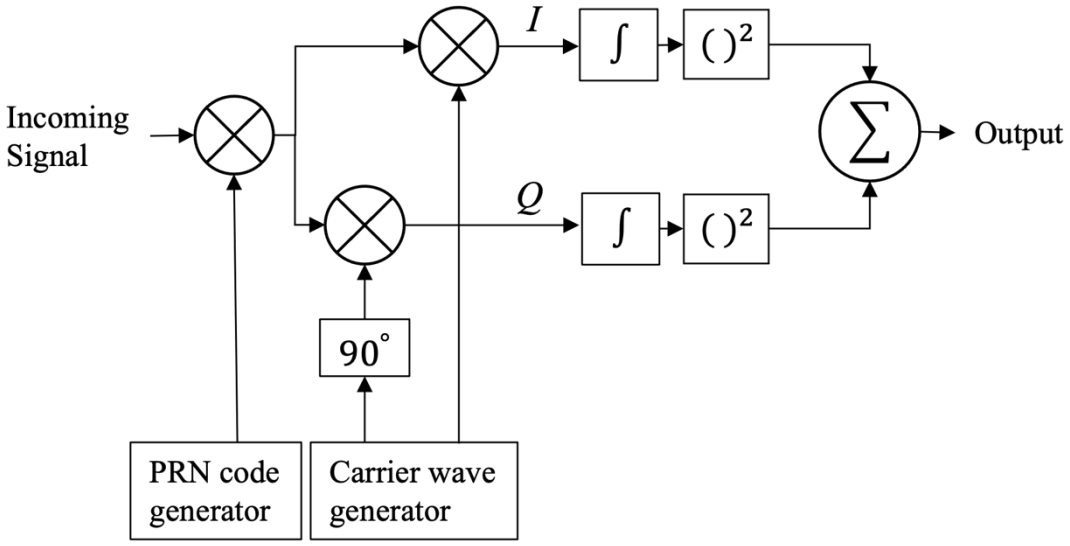

Figure 3. Serial Search Acquisition Block Diagram

2.2 Acquisition methods

Acquisition process described above can be approached in several different ways. Two common implementations are called Serial Search Acquisition and Parallel Search Acquisition. These methods are described in the upcoming sections.

2.2.1 Serial Search Acquisition

This essence of Serial Search Acquisition method is to iterate through all possible combinations of code phase and carrier frequency, until the correct values are found.

As seen from Figure 3 the incoming signal is first multiplied by the locally generated PRN sequence. Resulting signal is multiplied by a locally generated carrier wave and its 90^0 phase-shifted version which generates an in-phase (I) and quadrature (Q) signals respectively. The I and Q signals are integrated over 1ms (length of one C/A code), then squared and finally summed together. The output value is the correlation between 1ms of signal and the locally generated C/A code. If it exceeds a predefined threshold, the code phase and carrier frequency values used to generate local PRN sequence and carrier wave are close enough to the real ones (Borre, et al., 2007, p. 76).

Main downside of the Serial Search Acquisition is the large number of possible code phase and carrier frequency combinations which totals to 41,923.

$$\underbrace{1023}_{code\ phases} * \underbrace{\left(\frac{2 * 10000}{500} + 1\right)}_{frequencies} = \underbrace{41{,}923}_{combinations} \qquad Eq.\ 3$$

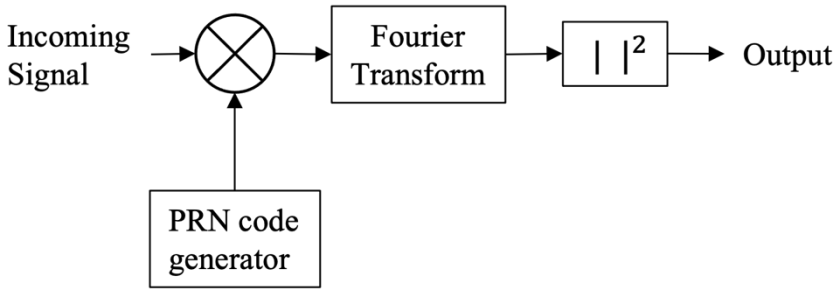

Figure 4. Parallel Frequency Space Search Acquisition.

Considering number of samples to be processed each millisecond ($f_s/1000$) the algorithm is computationally expensive and might bring about performance issues when considered in the context of real-time implementation (Borre, et al., 2007, p. 77).

On the bright side, Serial Search Acquisition method only requires multiplication and accumulation operations and can be implemented directly following the block diagram in Figure 3.

2.2.2 Parallel Search Acquisition

Sequentially searching through all possible code phases and frequencies is an $O(n^2)$ complexity operation. Eliminating or parallelizing search of code phase or carrier frequency would reduce complexity to $O(n)$ thus drastically increase the performance of the whole algorithm.

Parallel search acquisition does exactly that, by parallelizing search for one of the parameters.

2.2.2.1 Parallel Frequency Space Search Acquisition

As the name suggest this method parallelizes search of carrier frequency thus reducing the number of computations by a factor of 41.

As in the Serial Search method incoming signal is multiplied by a locally generated PRN sequence corresponding to a specific satellite with a specific code phase between 0 and 1022. Resulting signal is then transformed into frequency domain by a Fourier Transform (Figure 4). If the code phase of PRN sequence is correct the output of Fourier transform will show a distinct peak in magnitude. To find the peak frequency the absolute value of all components is calculated (Figure 4). Frequency corresponding to the peak index is the frequency of the carrier wave (Borre, et al., 2007, p. 78).

Resolution of the identified frequency will depend on the sampling frequency of the incoming signal. Since 1ms of data is processed, number of samples is found as $f_s\ /1000$. If sampling frequency is $f_s = 10MHz$ the number of samples is $N = 10{,}000$.

When using Discrete Fourier Transform with length of 10,000 the first $N\ /2$ output samples represent frequencies from 0 to $(\frac{f_s}{2} - 1)Hz$. That results in the frequency resolution output to be:

$$\Delta f = \frac{f_s/2}{N/2} = \frac{f_s}{N} \qquad \textit{Eq. 4}$$

Thus, with a sampling frequency $f_s = 10Mhz$ resulting frequency resolution is $1kHz$ (Serial Search Acquisition has a constant frequency resolution of $500Hz$).

Parallelizing frequency search reduces the search steps by a factor of 41 with a cost of a Fourier Transform on each iteration. Thus, performance of the Parallel Frequency Space Search is directly related to the efficiency of the Fourier transform algorithm. Fast Fourier Transform (FFT) is commonly utilized *(Schafer & Oppenheim,)*.

2.2.2.2 Parallel Code Phase Search Acquisition

Number of search bins for code phase is almost 25 times larger than it is for carrier frequency. Thus, it is only reasonable to suggest that parallelizing code phase search will yield a superior acquisition method. Parallel Code Phase Search Acquisition does exactly that.

Instead of multiplying the incoming signal by a PRN code with 1023 different variations of code phase, it is more convenient to run a circular cross correlation between the input and the PRN code without shifted phase. This process can be accomplished through Fourier Transforms and is depicted in the block diagram in Figure 5.

Incoming signal is multiplied by *I* and *Q* versions of locally generated carrier waves. Resulting complex signal is converted into frequency domain using Fourier transform. Locally generated PRN code is also transformed into frequency domain and complex conjugated after. Two signals are then multiplied and the result in converted into time domain through an Inverse Fourier Transform (IFT). Absolute value of the output of the IFT represents the correlation between the incoming signal and the PRN code. Distinct peak will be present in case when carrier frequency is close to the real frequency. Index

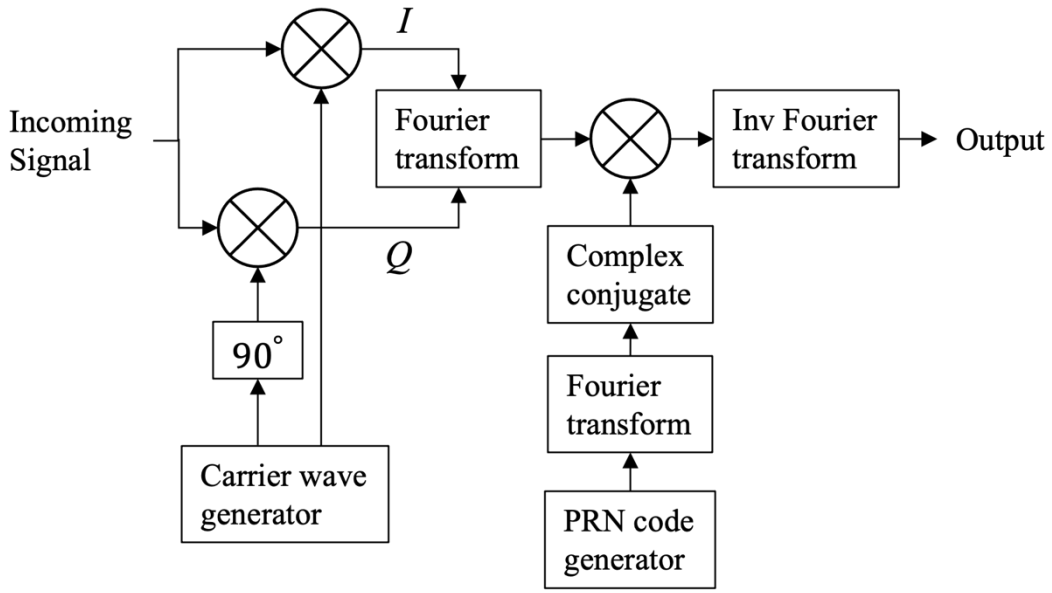

Figure 5. Parallel Code Phase Search Acquisition Block Diagram.

of that peak corresponds to PRN code phase of the incoming signal (Borre, et al., 2007, pp. 81-83).

Resulting code phase is more accurate when compared to other acquisition methods. Since the output of the IFT has number of samples $N = f_s/1000$ resulting code phase will have a value between 0 and N. Since minimum sampling frequency required to cover most of the GPS signal power is 4MHz, minimum value for $N = 4000$. Therefore, accuracy of the identified code phase will always be at least ~4 times greater than the reference accuracy of 1023 chips.

This method effectively trimmed number of search iterations to 41 per satellite. Since PRN code can be prepared off-line algorithm requires only one FFT and one IFFT per iteration. Thus, efficiency of Parallel Code Phase Search is closely coupled with the efficiency of FFT and IFT implementations.

3 CARRIER AND CODE TRACKING

Coarse values for code phase and carrier frequency are passed on to the tracking algorithm. Here these values are refined and kept track of. Using refined parameters incoming signal can be demodulated producing navigation bits in the output. This chapter provides brief overview on the algorithms used for code and carrier parameters tracking and details of its implementation in Gnu radio.

3.1 Demodulation

To transmit the low frequency navigation message in the UHF band it must be modulated onto a high frequency carrier wave (see section 1.1). One the receiver end, navigation message must be extracted from the carrier wave. This process is called demodulation and is described below.

Signal transmitted from a single satellite (Eq. 1), after the A/D conversion on the frontend can be depicted as:

$$s_k(n) = C_k(n)D_k(n)\,cos(\omega_{IF}n) + e(n) \qquad \textit{Eq. 5}$$

Where n has unit of $1/f_s$ s, indicating that the signal is discrete in time, ω_{IF} is the intermediate frequency, to which front end has down converted the signal and $\mathrm{D_k}$ is the navigation message (Borre, et al., 2007, p. 87). P code, present in the initial signal, is distorted by the narrow band-pass filter around the C/A code and is represented as noise $e(n)$.

Multiplying the signal in Eq. 5 by the exact replica of carrier cosine results in:

$$\begin{aligned} s_{\mathrm{k}}(n)\cos(\omega_{IF}n) &= C_{\mathrm{k}}(n)\mathrm{D}_{\mathrm{k}}(n)\cos(\omega_{IF}n)\cos(\omega_{IF}n) \\ &= \frac{1}{2}C_{\mathrm{k}}(n)D_{\mathrm{k}}(n) + \frac{1}{2}\cos(2\omega_{IF}n)\,C_{\mathrm{k}}(n)D_{\mathrm{k}}(n) \end{aligned} \qquad \textit{Eq. 6}$$

After the low pass filter the signal will look like:

$$\frac{1}{2}C_{\mathrm{k}}(n)D_{\mathrm{k}}(n) \qquad \textit{Eq. 7}$$

To remove the code C_{k}, signal is correlated with local code replica. The output is then:

$$\sum_{n=0}^{N-1} C_{\mathrm{k}}(n)C_{\mathrm{k}}(n)D_{\mathrm{k}}(n) = ND_{\mathrm{k}}(n) \qquad \textit{Eq. 8}$$

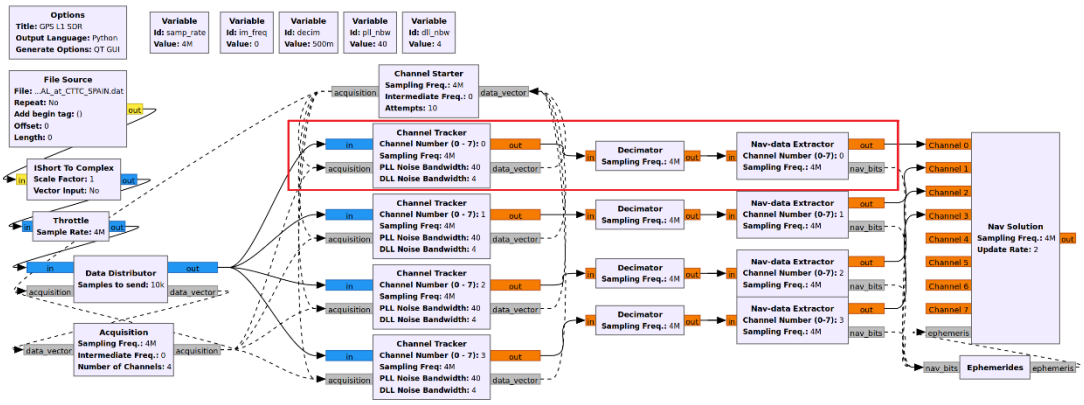

Figure 6. GPS L1 SDR flowgraph built in Gnu Radio Companion. Highlighted in read is a series of signal processing blocks, operating on a single satellite signal from the constellation commonly referred to as channel.

Where N is the number of samples in the signal (Borre, et al., 2007, p. 88).

3.2 Tracking

To retrieve navigation message D_k from the incoming signal exact values of carrier frequency and C/A code phase are required. Since satellites and often the receiver are in constant movement a frequency shift called doppler shift occurs. To preserve correct values of carrier frequency and code phase it is necessary to track this shift. GPS Tracking algorithm is designed to achieve this task. It consists of two parts: carrier wave tracking and C/A code tracking.

3.2.1 Carrier tracking

The task of carrier tracking algorithm is to keep track of carrier wave frequency and phase changes. This section describes the algorithm used to achieve it.

Majority of tracking algorithms are implemented using an analytical linear phase locked loop (PLL) model that can be used to predict performance, derived by (Ziemer, 1985).

Navigation bit transition requires the PLL implementation to be 180° phase shift insensitive. One such implementation is called Costas Loop. Its block diagram is depicted in Figure 7. Goal of the Costas Loop is to keep all energy of the signal in the I $(in-phase)$ arm. To achieve that, some sort of feedback to the oscillator is necessary.

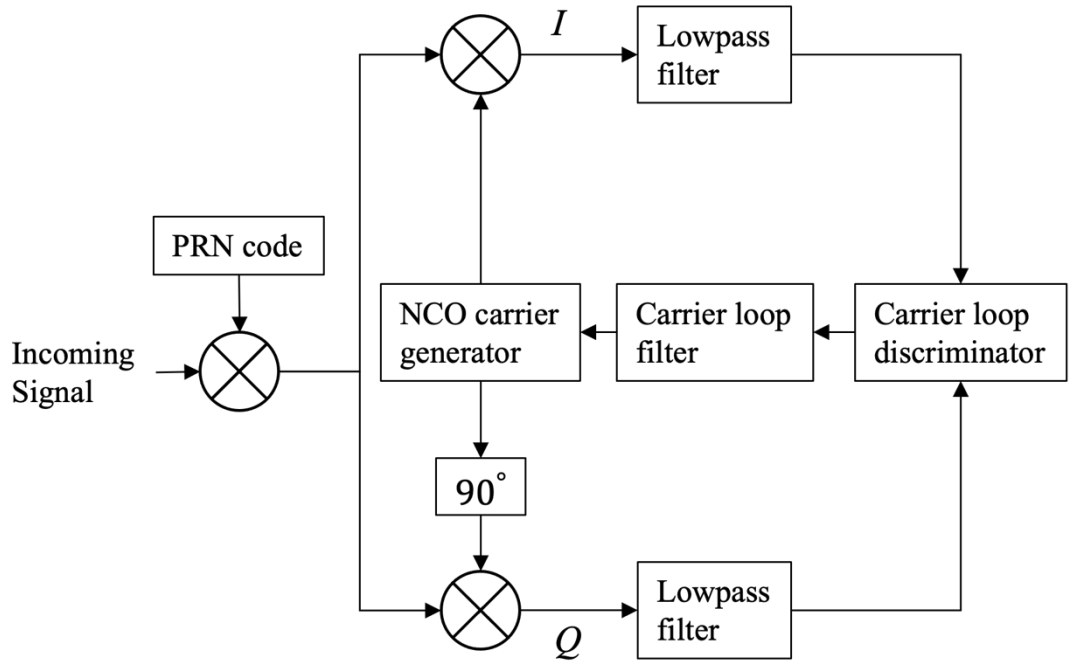

Figure 7. Costas Loop implementation block diagram (Borre, et al., 2007, p. 94)

When carrier wave replica phase is not perfectly aligned with the original, multiplying it and its 90° shifted version with the incoming signal will yield I and Q signals:

$$D_{\mathrm{k}}(n)cos(\omega_{IF}n + \varphi) = \frac{1}{2}D_{\mathrm{k}}(n)cos(\varphi) + \frac{1}{2}D_{\mathrm{k}}(n)cos(2\omega_{IF}n + \varphi) \qquad \textit{Eq. 9}$$

$$D_{\mathrm{k}}(n)sin(\omega_{IF}n + \varphi) = \frac{1}{2}D_{\mathrm{k}}(n)sin(\varphi) + \frac{1}{2}D_{\mathrm{k}}(n)sin(2\omega_{IF}n + \varphi) \qquad \textit{Eq. 10}$$

When I and Q signals are both lowpass filtered, following terms will remain:

$$I_{\mathrm{k}} = \frac{1}{2}D_{\mathrm{k}}(n)cos(\varphi) \qquad \textit{Eq. 11}$$

$$Q_{\mathrm{k}} = \frac{1}{2}D_{\mathrm{k}}(n)sin(\varphi) \qquad \textit{Eq. 12}$$

The phase error can be then found as:

$$\varphi = tan^{-1}(\frac{Q_{\mathrm{k}}}{I_{\mathrm{k}}}) \qquad \textit{Eq. 13}$$

This error is sent back as a feedback term to the carrier generator. Eq. 13 demonstrates, that the phase error is minimum when correlation in the quadrature-phase arm is zero, and correlation value in the in-phase arm is maximum. If the signal is tracked properly

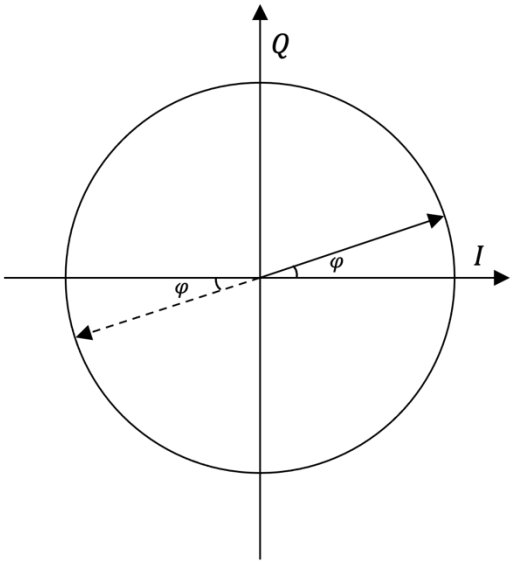

Figure 8. Phasor diagram showing the phase error between carrier wave replica and the input carrier wave (Borre, et al., 2007, p. 96).

sum of I_k and Q_k is aligned with the I axis (Figure 8), thus, when 180° phase shift occurs, nothing happens. This property of Costas Loop makes it a very common choice for the Carrier tracking algorithm in GPS signal processing (Borre, et al., 2007, p. 94).

3.2.2 Code tracking

Code tracking in a GPS receiver is usually implemented using delay locked loop (DLL) also known as early late tracking loop. This section provides basic understanding of this algorithm.

Main idea behind DLL is to correlate the incoming signal with early late and prompt (E, P and L) versions of C/A code, spaced by 0.5 chips. The code version which demonstrated better correlation results will thus dictate how the code phase should be adjusted. A case when prompt version has the best correlation and early and late versions demonstrate similar results, would mean that Code Phase is properly tracked (Borre, et al., 2007, p. 96).

As discussed in chapter 3.2.1, when a carrier code phase error is present signal power is distributed in both quadrature and in-phase arms. Thus, for DLL to be successful, it should also operate on both arms. Block diagram of such DLL implementation is depicted in Figure 9. The discriminator to find the feedback term in this case should consider early and late correlation results from both in-phase and quadrature arms. For noisy signals normalized early minus late power discriminator demonstrates great result (Borre, et al., 2007, p. 100). It is described as:

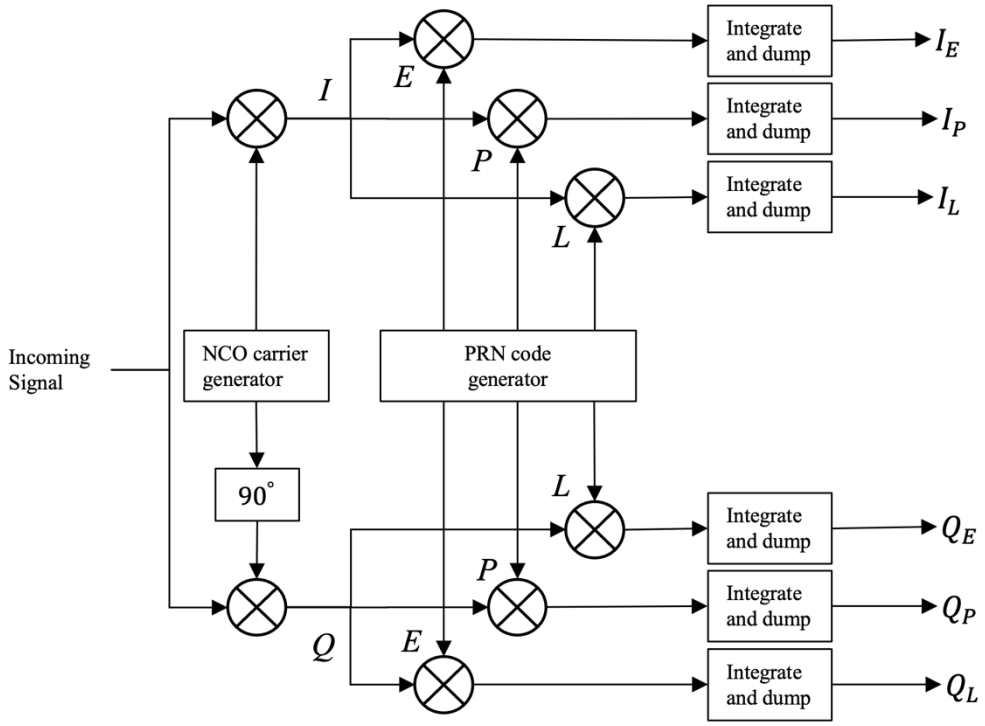

Figure 9. DLL block diagram with 6 correlators (Borre, et al., 2007, p. 97).

$$D = \frac{(I_E^2 + Q_E^2) - (I_L^2 + Q_L^2)}{(I_E^2 + Q_E^2) + (I_L^2 + Q_L^2)} \qquad \textit{Eq. 14}$$

Where I_E, Q_E, I_L and Q_L are outputs from four of the six correlators, shown in Figure 9 (Borre, et al., 2007). This discriminator is independent of the performance of carrier tracking loop since it accounts for both I and Q arms.

4 NAVIGATION DATA PROCESSING

Output from tracking after decimation is truncated value of in-phase arm correlated with full C/A code and integrated over one millisecond. The signal stream is thus a 50bps navigation data up sampled to 1000Hz. Using parameters encoded in navigation data along with signals travel time and time of arrival from at least 4 satellites is then enough to compute receiver's position.

4.1 Locating Subframe Start and Navigation-bit Transition

To synchronize data from all channels, start of a common subframe in each channel must be identified. The process of identifying this subframe in the incoming navigation data sequence is described below.

Once signal incoming from tracking is truncated to 1 and -1, we can proceed to identify subframe position and convert the signal to 50bps navigation data. As mentioned in section 1.1.1 each subframe consists of 300 bits, which are further divided into 10 30-bit words. First two words of every subframe are referred to as telemetry word (TLM) and hand over word (HOW). They contain all necessary information to locate a subframe in the signal. Their structure can be seen in Figure 10. Here 8-bit preamble in the beginning of TLM is used to locate the start of the subframe. Last 6 bits of both TLM and HOW are reserved for parity. It is used to check if the identified subframe is valid (Borre, et al., 2007, pp. 109-110).

Upconverting 50bps preamble to 1000Hz and cross-corelating it with the incoming signal allows us to identify possible subframe start candidates. Knowing that each subframe is 6 seconds or 300 bits long, we can further filter identified candidates to be exactly $300 * 20$ samples apart. Once position of such candidate is located in the signal, parity check is used to confirm its validity. Subframe start simultaneously marks the bit transition. Thus, once identified receiver can procced to extract the navigation data.

4.2 Extracting Navigation Data

Once a start of a subframe is identified navigation bits can be collected to extract all required satellite information. Navigation data is contained within ephemeris and almanac parameters encoded in one full frame. Thus, at least $300 * 5$ bits must be collected. Decoding of ephemeris and almanac parameters is done according to ICD-GPS-200 (Arinc Research Corporation, 1991). Ephemeris parameters are listed in Table 1.

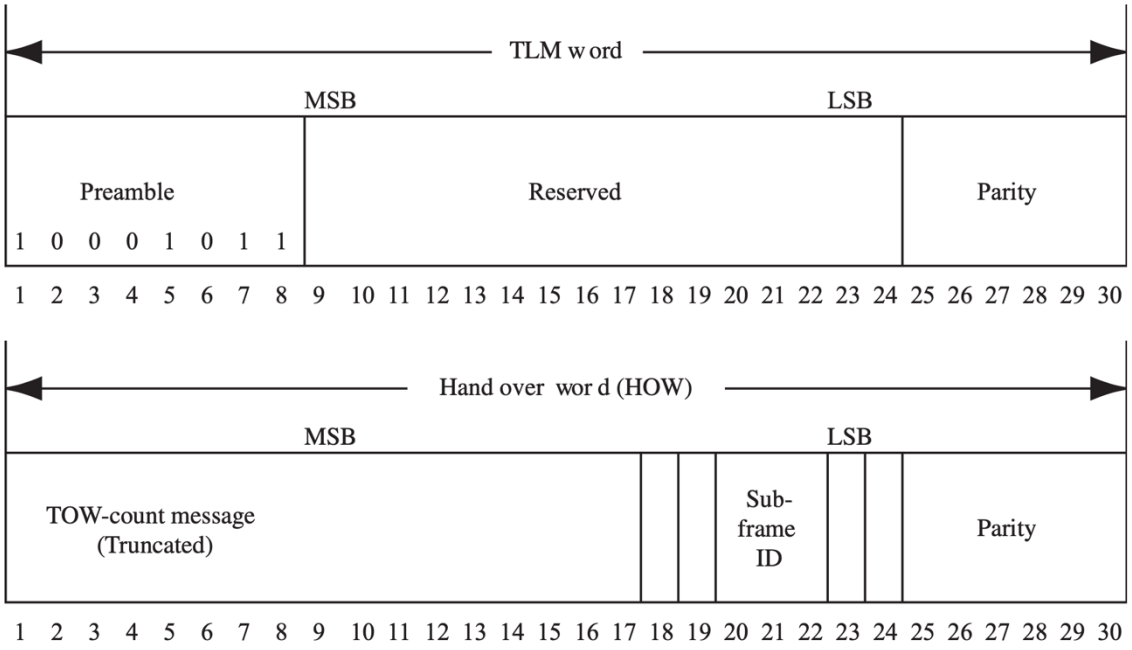

Figure 10. The first two words of each subframe, referred to as telemetry (TLM) word and hand over word (HOW) (Borre, et al., 2007, p. 113).

4.3 Time of Transmission and Arrival

Besides ephemeris and almanac parameters it is necessary to identify when a subframe was transmitted from a satellite and when it was received on the receiver. This section describes the process to obtain this information.

Every HOW of a subframe contains a truncated version of time of the week (TOW) (see Figure 10). It is often referred to as Z-count, which is the number of seconds passed since the last GPS week rollover in units of 1.5 seconds. This rollover happens at midnight from Saturday to Sunday (Borre, et al., 2007). Since $604{,}800s/1.5s = 403{,}200s$ maximum value of the Z-count is 403,199. The Z-count in HOW is a truncated version containing only 17 most significant bits (MSB). This makes the Z-count in HOW increase in steps of 6s, which corresponds to time between transmission of two consecutive subframes (Borre, et al., 2007, pp. 112-114). The truncated Z-count in HOW corresponds to time when the next subframe was transmitted. Thus, the transmission time of current subframe in seconds would be:

$$TOW = Z_{count} * 6 - 6 \qquad \textit{Eq. 15}$$

IODE	Issue of data, ephemeris
Δn	Mean motion correction
μ_0	Mean anomaly at t_{0e}
e	Eccentricity
$\sqrt{a}$	Square root of semi-major axis
t_{0e}	Reference epoch of ephemeris
Ω_0	Longitude of ascending node at t_{0e}
i_0	Inclination at t_{0e}
ω	Argument of perigee
$\dot{\Omega}$	Rate of Ω_0
$\dot{i}$	Rate of i
C_{rs}, C_{rc}	Correction coefficient of sine and cosine terms of r
C_{is}, C_{ic}	Correction coefficient of sine and cosine terms of i
C_{us}, C_{uc}	Correction coefficient of sine and cosine terms of ω

Table 1. Ephemeris Parameters (Borre, et al., 2007, p. 114).

Time of arrival of a subframe on the receiver side can be precisely calculated from absolute sample count from the start of the receiver till the start of the subframe.

Knowing the TOW of when a subframe was transmitted and when it was arrived on the receiver together with ephemeris and almanac parameters from at least 4 satellites allows us to calculate pseudoranges from the receiver to each satellite and the position of each satellite in the sky. This information is used to calculate receiver's position.

4.4 Position Computation

After precise values for pseudoranges and satellites' position are calculated, computation of receiver's position is possible. This computation is expectedly very mathematical and uses advanced trigonometry and linear algebra. The results are the Earth-centered and Earth-fixed (ECEF) coordinates X, Y and Z which can be further

converted in any other desired coordinate system. The theory behind computation of GPS position is described in detail by (Borre, et al., 2007, pp. 121-125).

5 RECEIVER IMPLEMENTATION

5.1 Top Level Architecture

The top level of the receiver consists of GPS signal source, acquisition system, tracking system and navigation data processing system. The overarching architecture design is depicted on Figure 11.

The project is implemented as an out of tree (OOT) GR module containing signal processing blocks necessary to implement each system, mentioned above. Every block is in essence a C++ Class with a header and implementation file. Further details on the OOT modules can be found on GR OOT wiki page (Wiki GNU Radio, 2021).

An effort was made to keep encapsulation at a reasonably low level, to preserve code readability and encourage further development.

5.2 Acquisition System

Acquisition process is implemented through 3 custom GR blocks: “Data Distributor”, “Acquisition” and “Channel Starter” (see Figure 12). Since the process involves expensive computations, acquisition is done asynchronously to the main signal data flow, with the use of GR messages (see section 1.3.2). Implementation details of each block are described in the following sections.

5.2.1 Data Distributor

[data_distributor_impl.h, data_distributor_impl.cc]

“Data Distributor” receives signal data stream directly from the GPS signal source and is used to provide signal data for acquisition process, when required. Acquisition is a computationally demanding process. Therefore, to decouple it from the main signal stream, all acquisition related information is distributed using GR messages (see section 1.3.2). This effectively makes acquisition an asynchronous process.

Data Distributor block passes the signal stream through and delivers signal recording for the acquisition upon request. The distribution of data is implemented through a pair of stream and message ports (see Figure 12).

Stream ports are used to path through the main signal stream. Output message port “data_vector” is used to send out the collected samples. Input message port “acquisition” is used to trigger the acquisition process. This is necessary when a

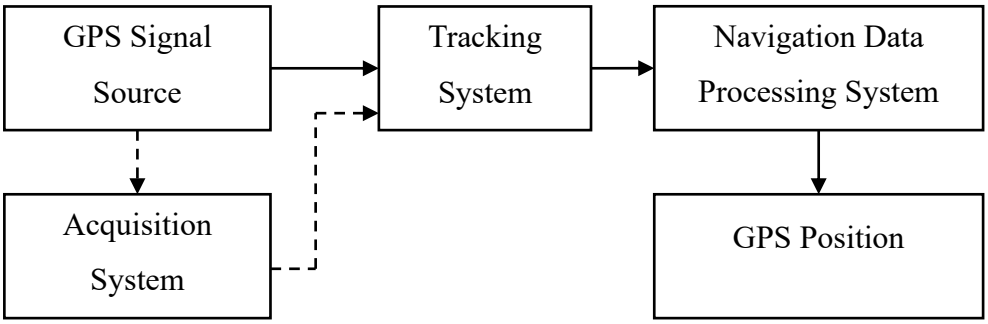

Figure 11. Top level receiver architecture design. Solid lines represent continuous signal stream. Dotted lines represent asynchronous messages.

reacquisition is required. An example would be a satellite assigned to a channel going out of sight, or acquisition process failing to identify required number of satellites.

"Data Distributor" block has a single parameter "Samples to send" which defines how many samples is going to be collected. In theory 1ms snapshot of the signal should suffice for acquisition, however this snapshot might capture a moment where navigation bit transitions from +1 to -1 or vice versa. This would mean that part of the C/A code contained within the snapshot is negated when the other part is not. To mitigate this issue Acquisition requires at least 2ms worth of signal data (Borre, et al., 2007, p. 85). Therefore, value of "Samples to send" defaults to $(samp\_rate * 2.5)/1000$.

5.2.2 Acquisition

[acquisition_impl.h, acquisition_impl.cc]

"Acquisition" block (see Figure 12) is responsible for identifying visible satellites and providing this information to Tracking System (section 5.3.1).

Acquisition process is initiated the moment it receives data on the "data_vector" input message port. Acquisition is implemented using the Parallel Code Phase Search method (chapter 2.2.2.2).

Search algorithm iterates through all 32 known PRNs and records an entry, for each satellite with signal strength above a predefined threshold. This entry contains the PRN number and the channel number this satellite will be assigned to. When the process is complete, results are sorted by signal strength and sent out sequentially via "acquisition" output message port.

5.2.3 Channel Starter

[channel_starte_impl.h, channel_starter_impl.cc]

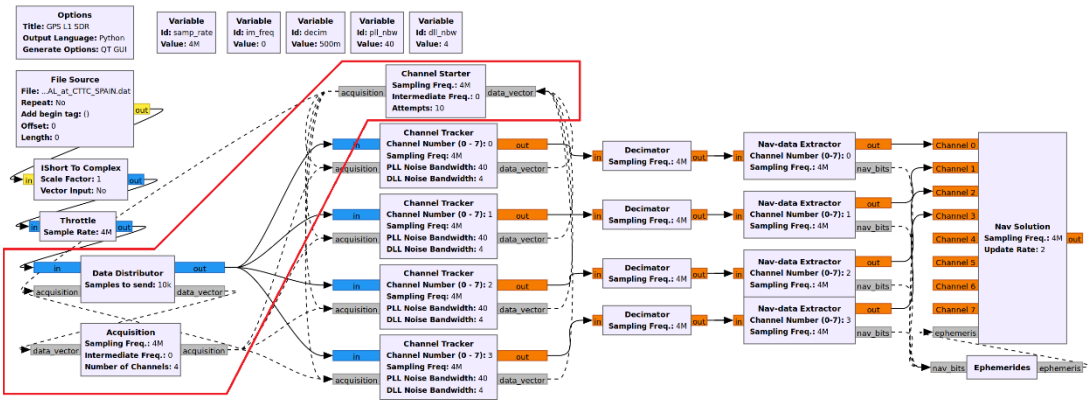

Figure 12. GPS L1 SDR flowgraph built in Gnu Radio Companion. Highlighted in red are signal processing blocks responsible for acquisition process.

To successfully track the Code Phase and Carrier Frequency, it is important to provide the tracking algorithm correct initial carrier frequency and code phase values. As mentioned in section 2.2.2.2 Parallel Code Phase Search acquisition method does exactly that. However, if carrier frequency resolution of 500Hz is enough to identify a visible satellite, it might be insufficient for the tracking algorithm to converge. Thus, a fine resolution frequency search is required. This section describes the implementation details of this algorithm in the Channel Starter block (see Figure 12).

Since the exact value of Code Phase is identified during the Parallel Code Phase Search, signal can be perfectly aligned with the start of the spreading sequence. C/A code can then be removed from signal. Converting the resulting signal to the frequency domain will expose the carrier frequency. In this case a greater number of samples results in an improved spectral resolution. “Channel Starter” block is designed to use identified code phase and perform fine frequency search to ensure tracking algorithm success.

Block expects 11ms long signal snapshot and its associated PRN value as a PMT message on the “data_vector” input message port. Once Channel Starter receives the data, Parallel Code Phase search algorithm is executed using the first 2ms, to find coarse values for Code Phase. Once code phase value is obtained, the snapshot is aligned with the start of the C/A code. Then DC bias is removed from the complete 10ms snapshot of the signal. The snapshot is then multiplied by locally generated C/A code and transformed into frequency domain using Fourier transform with the number of points equal to:

$$N_{fft} = 8 * 2^{ceil(log_2 N_{signal})} \qquad Eq.\ 16$$

Where N_{signal} is the number of samples in 10ms signal snapshot. High number of FFT points results in better frequency resolution. Absolute values of the FFT are then used to detect a peek, which indicates the carrier wave presence. Index of the peek is used to calculate the carrier wave (see Figure 15).

Once identified, carrier frequency and code phase are broadcasted through the acquisition message port.

"Channel Starter" assigns each PRN value a limited number of attempts, value for which is provided by the "Attempts" input parameter of the block. Once the number of attempts for any PRN is exhausted without identifying the coarse values for carrier frequency and code phase, "Channel Starter" notifies "Data Distributor" via "acquisition" message port, that reacquisition is required.

5.3 Tracking System

Carrier and Code tracking is a computationally expensive operation, which in the context of real time processing must be both efficient and reliable. This section describes the implementation details of tracking system in the project.

When discussing performance, carrier frequency and code phase tracking are two most demanding operations in a software-defined GPS receiver. This is dictated by the fact, that they must operate on the full range of samples from ADC, meaning that the data rate is equal to f_s. Thus, when designing a tracking block for a real time receiver, performance metrics should be one of the top priorities.

Both Carrier and Code Phase tracking were implemented in "Channel Tracker" block. Output of the tracking block is then decimated by "Decimator" block from f_s to 1000Hz, so that each sample represents 1ms of the signal (see Figure 13).

5.3.1 Channel Tracker

[tracking_impl.h, tracking_impl.cc]

Channel tracker block is the first part of a GPS receiver channel (see Figure 6) which extracts and outputs navigation bits. Intricacies of its implementation are given below.

Channel Tracker receives information in two different ways from three different sources. The incoming signal is received on a single input stream port at a frequency

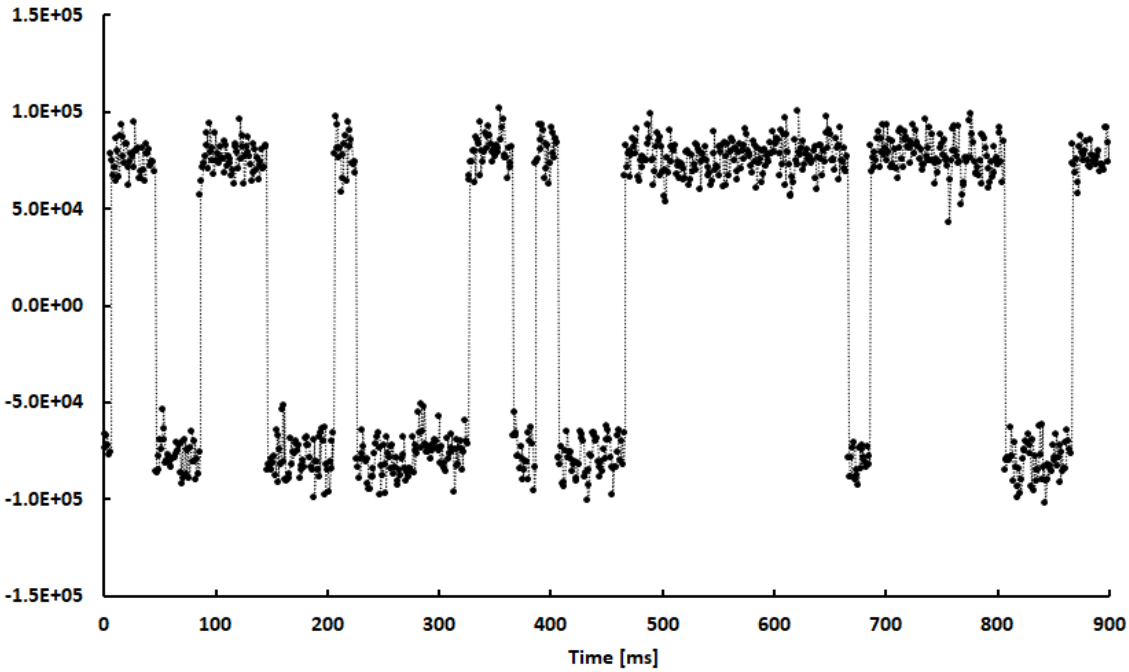

Figure 13. Output of Decimator block before quantization. Each data point represents a correlation result of 1ms of incoming signal with the C/A code. The greater the absolute value of the correlation, the stronger the signal is. Bit transition occurs every 20ms.

equal to f_s. All acquisition related information is received on the "acquisition" message port. Acquisition messages are constructed using PMT pairs (Wiki GNU Radio, 2021) and are processed accordingly using the key parameter of the pair as an identifier. Therefore, a key of a PMT pair will dictate whether message contains acquisition results, carrier frequency and code phase from the channel starter, or a reacquisition request for the Acquisition block.

Once PRN value is received from the Acquisition a process for acquiring coarse values for Carrier Frequency and Code Phase can be initiated.

5.3.1.1 Coarse Carrier Frequency and Code Phase Acquisition

Channel Tracker sends 11ms worth of incoming signal to "Channel Starter" block. Once channel starter identified carrier frequency and code phase, they are sent via acquisition message port. Since "Channel Starter" is working asynchronously it is important to adjust the received code phase according to the time delay between when the data was collected, and the code phase was received. Since 1ms of incoming signal is known to contain approximately $f_s/1000$ samples, the code phase at the moment of reception can be found as:

$$\tau_{exact} = blksize - ((N_{total} - \tau_{estimate}) \,\%\, blksize) \qquad Eq.\ 17$$

Where $\tau_{estimate}$ is the value of the Code Phase estimated by the "Channel Starter", N_{total} is the total amount of samples processed from the start of collecting 11ms of data to the reception of estimated code phase and $blksize$ is the ideal number of samples in 1ms of incoming signal.

5.3.1.2 PLL and DLL implementation specifics

After reference carrier wave frequency and C/A code phase are received and the incoming signal is aligned based on τ_{exact} (Eq. 17) tracking algorithm can start.

Gnu Radio sources the incoming signal to the blocks in batches, size of which is determined by the GR scheduler, and is not constant. Once the batch is received a function is called where tracking algorithm gets access to the batch samples array. On each iteration i through the sample array, values for I_{E_i}, I_{P_i}, I_{L_i}, Q_{E_i}, Q_{P_i} and Q_{L_i} are calculated. To find correlation result over 1ms of the signal, new values of in-phase and quadrature early, prompt, and late signals are accumulated. Once number of iterations reaches the block size (number of samples contained in 1ms of incoming signal) accumulated value of I_P is provided to the output. Then, discriminators for both PLL and DLL are calculated. These values are used to update carrier frequency and phase, Code phase and to recalculate the block size. I_E, I_P, I_L, Q_E, Q_P and Q_L as well as internal iterator are reset to zero and processing of next millisecond with updated parameters starts.

Processed 1ms of the signal represents one navigation bit. Therefore, the logical output rate of the "Channel Tracker" block is 1000Hz. To correctly decimate the stream at the output, a full millisecond block must be processed within one function call, i.e., stream batch size should be equal to block size. This, however, is not possible due to the variable block size. Therefore, channel tracker block cannot decimate the signal and its effective output rate equals to f_s, with only 1000 logical values generated per second. The rest of the output values are set to 0.

5.3.1.3 Performance optimizations

Signal tracking is the most computationally expensive step in the GPS signal processing. Thus, optimization of the algorithms used for tracking is an important step in its implementation.

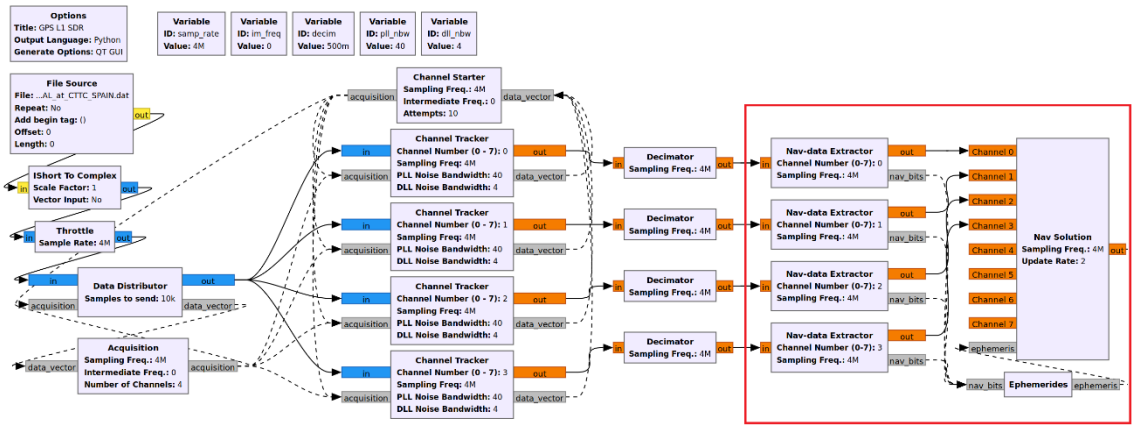

Figure 14. GPS L1 SDR flowgraph built in Gnu Radio Companion. Highlighted in red are signal processing blocks responsible for position computation.

Both PLL and DLL implementation coupled with demodulation contains only two computationally expensive operations. These include calculating PLL discriminator and computing sine and cosine for in-phase and quadrature carrier wave replicas.

Since PLL discriminator is only calculated once a millisecond, it doesn't create a significant performance bottleneck. New values for carrier sine and cosine however are recalculated on every iteration. Thus, implementing a fast and accurate method to calculate $sin(\theta t + \varphi)$ and $cos(\theta t + \varphi)$ is vital for performance metrics.

There are several approaches to this problem with one of the most common ones being a lookup table. However, a more precise solution with the use of rotation matrix was implemented. From Eq. 18, when $t = 0$ knowing sine and cosine for θ and φ, allows us to easily compute sine and cosine for the rest of required iterations by solving the matrix. Since t ranges from 0 to $blksize$, the need to call computationally expensive sine and cosine functions is reduced by a factor of $f_s/1000$:

$$\begin{bmatrix} cos(\theta(t+1) + \varphi) \\ sin(\theta(t+1) + \varphi) \end{bmatrix} = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} cos(\theta t + \varphi) \\ cos(\theta t + \varphi) \end{bmatrix} \qquad \textit{Eq. 18}$$

5.3.2 Decimator

As mentioned in section 5.3.1.2, the output rate of "Channel Tracker" is equal to f_s, when the rational output is only 1000Hz. Thus, it is necessary to decimate the stream, before sourcing it to other blocks.

"Decimator" block receives the stream from the Channel Tracker (see Figure 6) and decimates the incoming stream to 1000Hz. Each sample then represents 1ms of navigation data. Thus, a decimation factor of $f_s/1000$ is applied. Block fishes out the

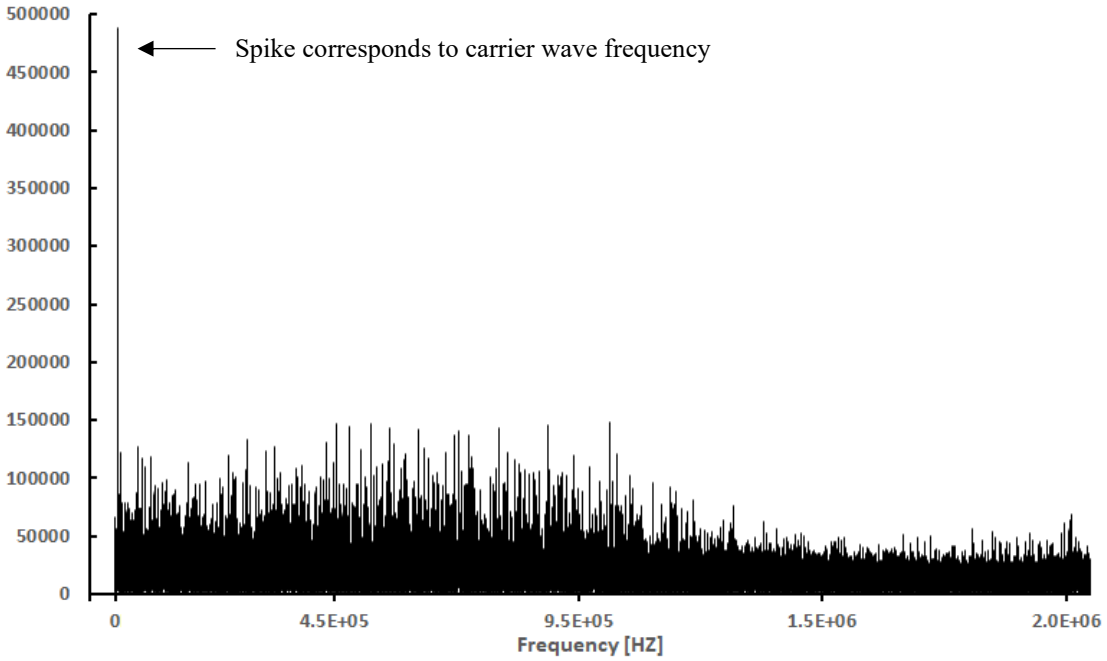

Figure 15.10 milliseconds of incoming signal after 0DC conversion, C/A Code removal and Fourier transform. Peek corresponds to carrier wave frequency.

sample with the I_P value and calculates the total number of samples processed up to that point. This value is saved in a stream tag and assigned to the corresponding output sample. The value of output sample is the I_P quantized to 1 and -1.

5.4 Navigation Data Processing System

Once navigation-bits are obtained from tracking processing of navigation data starts. The process is implemented in 3 signal processing blocks. “Nav-data Extractor” is responsible for providing data to “Ephemerides” block to decode the navigation frame. “Nav-Solution” block calculates pseudoranges and satellites’ position, which are then used to compute receiver’s position in ECEF and geodetic coordinates.

5.4.1 Nav-data Extractor

[nav_decoding_impl.h, nav_decoding_impl.cc]

“Nav-data Extractor” is the last block in the GPS receiver channel, i.e., it is the last block operating on a signal from a single satellite. Navigation bits stream, received on the input is directly passed through to the output. When the signal is properly tracked each incoming sample is expected to be tagged. This tag is expected to contain PRN number of the channel, which is used to clarify the source of the received navigation bits.

Main objective of the block is to collect uninterrupted sequence of navigation data enough to decode one full navigation frame. Thus, a starting point of any subframe in the incoming signal must be identified.

5.4.1.1 Locating Start of a Subframe

Locating start of a subframe is a straightforward operation which involves cross correlation of incoming signal with a known 8-bit preamble sequence.

Correlation is implemented by convolving navigation bits with the reverse version of preamble. Since navigation bits are arriving at 1000Hz, 50Hz preamble sequence must be up sampled to 1000Hz. Cross correlation will provide maximum correlation where preamble matches a sequence in the incoming navigation bits. However, because of Costas Loop's ability to track the signal with 180° phase shift (see Figure 16), correlation minimum can also occur, which would suppose a match with the negated version of a preamble (Borre, et al., 2007, pp. 110-111).

After subframe start candidates are identified, parity check on the respective subframe is conducted. Upon success of the parity check subframe start is confirmed.

5.4.1.2 Decoding Navigation Data

As described in section 4.2 exactly 1500 navigation bits are required to decode all ephemeris and almanac parameters transmitted from a satellite. With 50 bps data rate satellite needs at least 30 seconds to transmit a complete frame. Thus, once the subframe start is identified, receiver waits until $SubframeStart + 1500 * 20\ samples$ are collected in an array.

Once required number of samples are collected, array is down converted to 50 bps and sent via "nav_bits" message port to the "Ephemerides" block for encoding.

5.4.2 Ephemerides

[ephemerides_impl.h, ephemerides_impl.cc]

Like "Acquisition", "Ephemeris" block is also implemented asynchronously, i.e., all data transfer happens via message passing. This provides visual feedback on how Ephemeris parameters are distributed on the flowgraph.

Decoding process is straight forward and consists of reading the bits according to Table 2. All ephemeris parameters, along with the channel number it belongs to, are then sent on "ephemeris" message port.

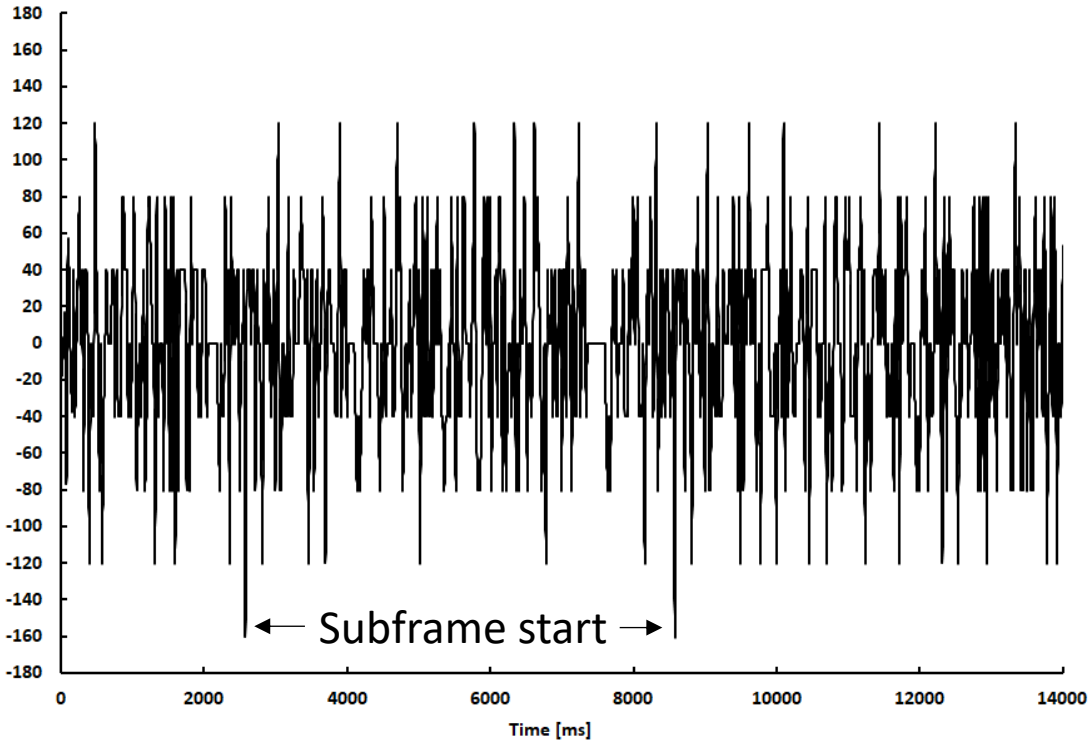

Figure 16. Result of cross-correlation between 14 seconds of navigation bits with preamble, up sampled to 1000Hz. Largest correlation result is negative indicating that the located preamble is inverted.

5.4.3 Nav-Solution

[nav_solution_impl.h, nav_solution_impl.cc]

"Nav-Solution" block was designed with 8 input stream ports, thus, can receive navigation data bits from up to 8 channels. The task of the Nav-Solution block is to calculate receiver's position with a certain frequency. Implementation details are described below.

5.4.3.1 Pseudoranges and Satellites' position

As described in section 4.3 to calculate pseudoranges receiver needs to know when the satellite transmitted the signal and when it was received on the receiver.

To synchronize time of arrival from all satellites, Nav-Solution finds the start of the same subframe in all the incoming channels. The procedure to find subframe start is the same as described in section 5.4.1.1. Synchronization is done by confirming that all identified subframes have the same TOW. Naturally time of arrival of this subframe will be slightly different in every satellite and can be calculated using absolute number of samples processed by the receiver at the moment when the bit indicating the start of this

Parameter	Number of bits	Scale Factor (LSB)	Unit
$IODE$	8		
C_{rs}	16*	2^{-5}	m
Δn	16*	2^{-43}	Semicircle/s
μ_0	32*	2^{-31}	Semicircle
C_{uc}	16*	2^{-29}	Radian
e	32	2^{-33}	dimensionless
C_{us}	16*	2^{-29}	Radian
$\sqrt{a}$	32	2^{-19}	$m^{1/2}$
t_{oe}	16	2^{4}	s
Ω_0	32*	2^{-31}	Semicircle
C_{is}	16*	2^{-29}	Radian
i_0	32*	2^{-31}	Semicircle
C_{rc}	16*	2^{-5}	m
ω	32*	2^{-31}	Semicircle/s
$\dot{\Omega}$	24*	2^{-43}	Semicircle/s
$\dot{i}$	14*	2^{-43}	Semicircle/s

Table 2. Decoding scheme for GPS ephemeris parameters. n means that the current n bits should be decoding using the two's-compliment (Borre, et al., 2007, p. 113).*

subframe was produced. This value is attached to the bit via a tag in the decimator block (section 5.3.2). Knowing the number of samples time of arrival can be calculated as:

$$T_{arrival} = \frac{N_{abs}}{f_s/1000} \qquad \textit{Eq. 19}$$

Where N_{abs} is the absolute sample number and f_s is the sampling frequency. Time of arrival of each channel together with the TOW of the subframe can then be tracked in real time and used to calculate pseudoranges.

Knowing the TOW and ephemeris + almanac parameters, receiver can keep track of the satellites' position.

5.4.3.2 Receiver Position calculation

Nav-solution block can calculate new pseudoranges and satellites' position values 1000 times per second. Reasonably, receiver's position can also be updated with the same

rate. Thus, update rate parameter of the Nav-Solution block accepts numbers from 1 to 1000 and dictates how many times per second receiver will recalculate its position.

All computations were based on MATLAB-based GPS Receiver contained in (Borre, et al., 2007). Linear algebra, matrix operations, numerical solvers and related algorithms were implemented using Eigen3 high-level C++ template library (Gaël Guennebaud, 2010).

Resulting position can be printed in ECEF or geodetic coordinates in the terminal.

6 CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

The designed software defined GPS receiver is structured to process signals streamed in real time either by a RF front end or from a GPS signal recording file.

Flowgraph based on the developed out of tree Gnu Radio module can compute position using 4 or more satellites in real-time. Utilizing Code Phase Frequency search algorithm with high efficiency FFT results in a fast acquisition process. Combining it with tracking optimizations and Gnu Radio's multithreaded design results in good performance metrics, which allows to process signals at 38.192MHz sampling rate on a civilian grade CPU. Obtained position coordinates demonstrate equal precision compared to post processing results of MATLAB-based GPS Receiver contained in (Borre, et al., 2007). Functional testing of the receiver is described in greater details in appendix A.

6.2 Future Work

Position computation algorithm described in (Borre, et al., 2007) doesn't demonstrate good precision when dealing with low sampling rate signals. Therefore, receiver's precision can be improved by incorporating a dedicated GNSS positioning library, RTKLIB (RTKLIB, 2013). This library is also used in GNSS-SDR project (Fernandez, et al., 2011).

To create efficient collaboration work environment for the project it is necessary to test every functional element in the receiver. GR actively promotes test driven development and necessary testing infrastructure is already present in the software. (Wiki GNU Radio, 2021).

In the scope of this thesis real time data stream was simulated in GR using a GPS signal recording file and a throttle block, which feeds samples from the source file with a delay, calculated from the provided sampling frequency. Naturally, the next step is to set up a RF front end with a GPS L1 antenna as a signal source to test and further fine tune the receiver.

Currently output can only be printed in the terminal using C++ Standard output stream. In future receiver should be able to save all obtained information, such as tracking performance, ephemeris parameters and position data in separate files for further analysis.

A. REAL TIME GPS RECEIVER, FUNCTION TESTING

A.1 General Outline

Receiver was tested on Linux Ubuntu 21.10 distribution, with an AMD Ryzen 7 2700X CPU. Two GPS L1 signal recordings were used as the signal source (see Table A. 1). Each signal was processed using a 6-channel flowgraph design depicted in Figure A. 1 and Figure A. 2.

File Name	GPSdata-DiscreteComponents-fs38_192-if9_55.bin	2013_04_04_GNSS_SIGNAL_at_CTTC_SPAIN.dat
Source	(Borre, et al., 2007)	(Fernandez, et al., 2011)
Sampling Freq.	38.192MHz	4MHz
IF	9.548MHz	0Hz
Data Type	8-bit integer	16-bit integer
Signal Type	Real valued signal	Interleaved I/Q signal
Duration	~50s	~100s

Table A. 1. GPS signal recordings used to test the receiver.

A.2 Tests execution

Acquisition system managed to identify 6 satellites in both signals. Snapshot of tracking performance is depicted in Figure A. 4 and Figure A. 3. Whilst processing the signal with 38.192MHz sampling rate CPU load was averaging at 50% (Figure A. 3). Signal with 4MHz sampling rate naturally requires less computation resources and was averaging at 20% CPU load (Figure A. 6).

Position for both signals was re-calculated twice a second. Results mapped in Google Maps can be found on Figure A. 8 and Figure A. 7. As expected, when processing 38.192MHz signal, GR receiver demonstrates identical results to what is obtained by the post-processing in MATLAB-based GPS Receiver. However, when processing the 4MHz signal, position computation algorithm used in the receiver results in a ~50m error, when compared to results from GNSS-SDR (Fernandez, et al., 2011) which makes use of RTKLIB (RTKLIB, 2013) library to compute position.

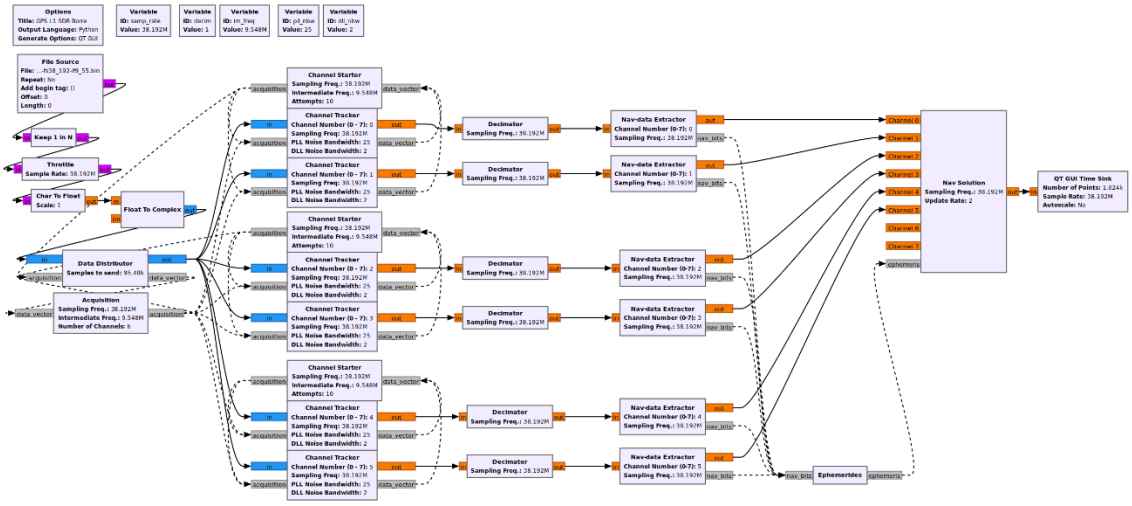

Figure A. 1. GR flowgraph used to process the "GPSdata-DiscreteComponents-fs38_192-if9_55.bin" signal. Real valued signal is converted to a 32-bit complex signal with 0 imaginary part.

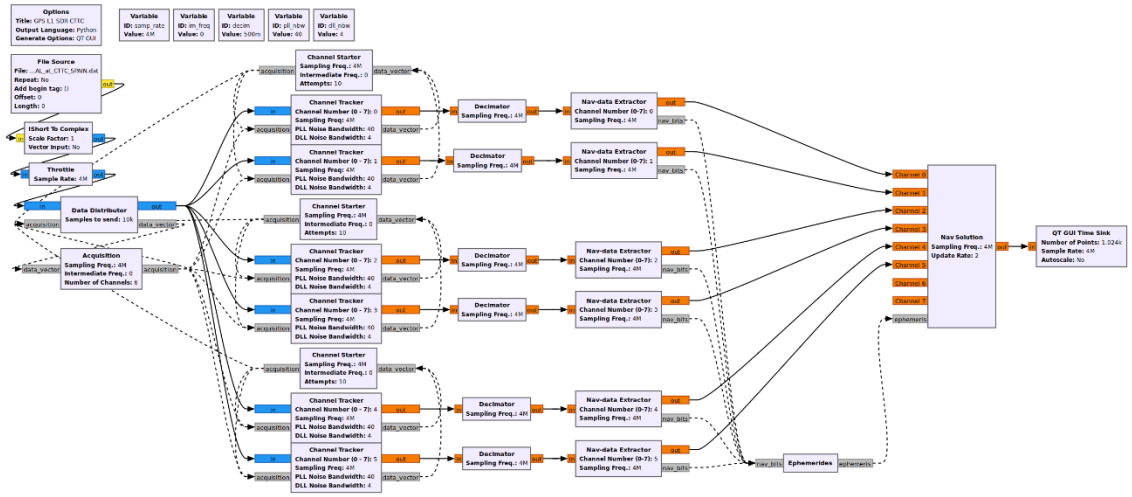

Figure A. 2. GR flowgraph used to process the "2013_04_04_GNSS_SIGNAL_at_CTTC_SPAIN.dat" signal. 16-bit interleaved I/Q signal is converted to 32-bit complex signal.

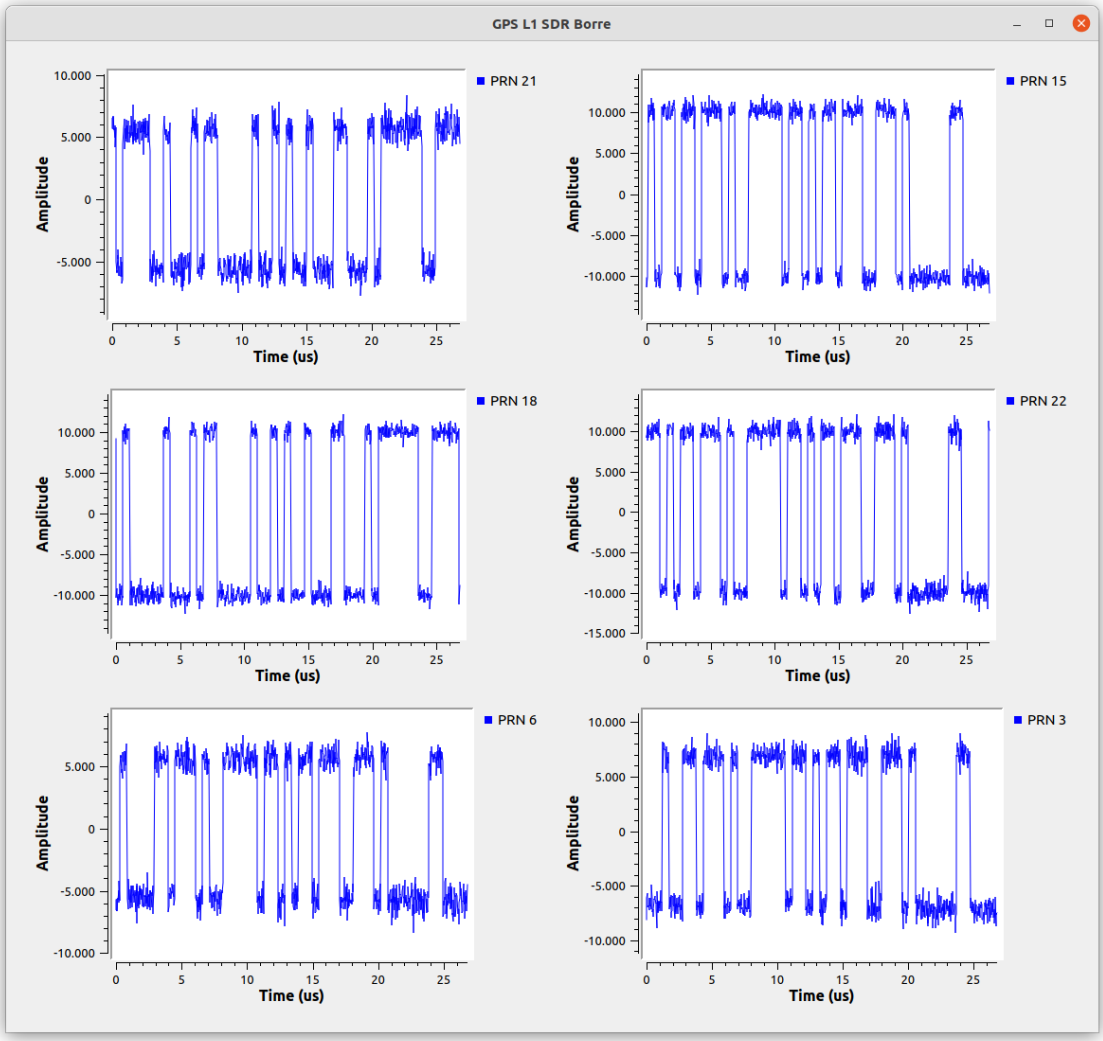

Figure A. 4. Snapshot of tracking results of the "GPSdata-DiscreteComponents-fs38_192-if9_55.bin" signal before truncation.

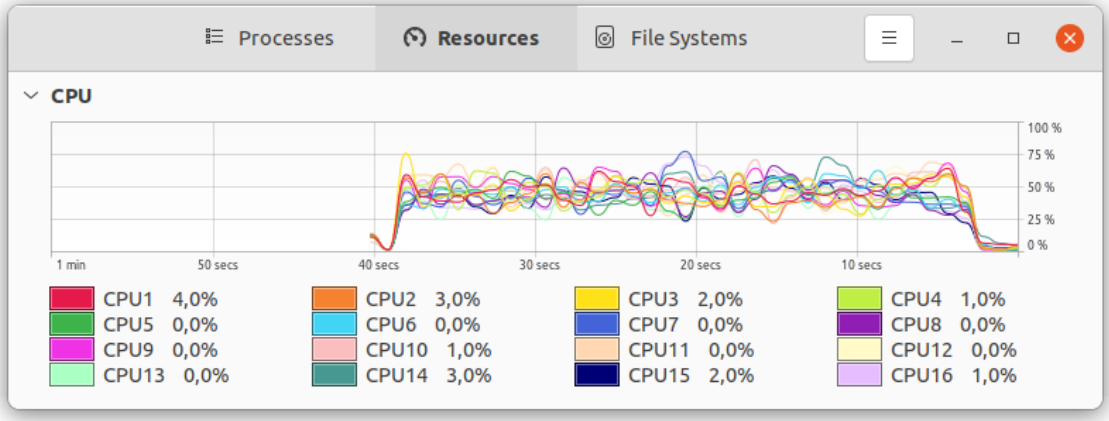

Figure A. 3. CPU load while processing "GPSdata-DiscreteComponents-fs38_192-if9_55.bin" signal.

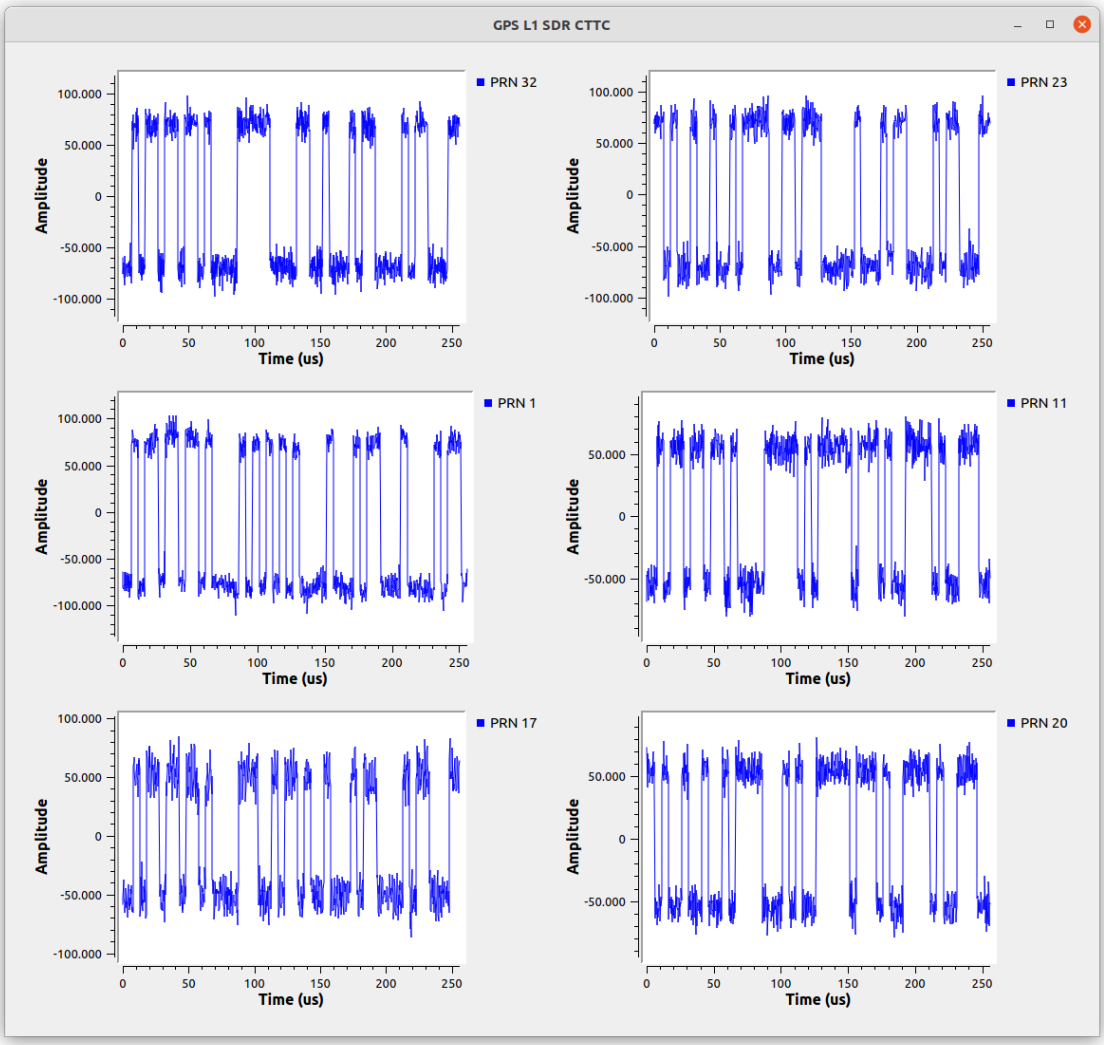

Figure A. 5. Snapshot of tracking results of the "2013_04_04_GNSS_SIGNAL_at_CTTC_SPAIN.dat" signal before truncation.

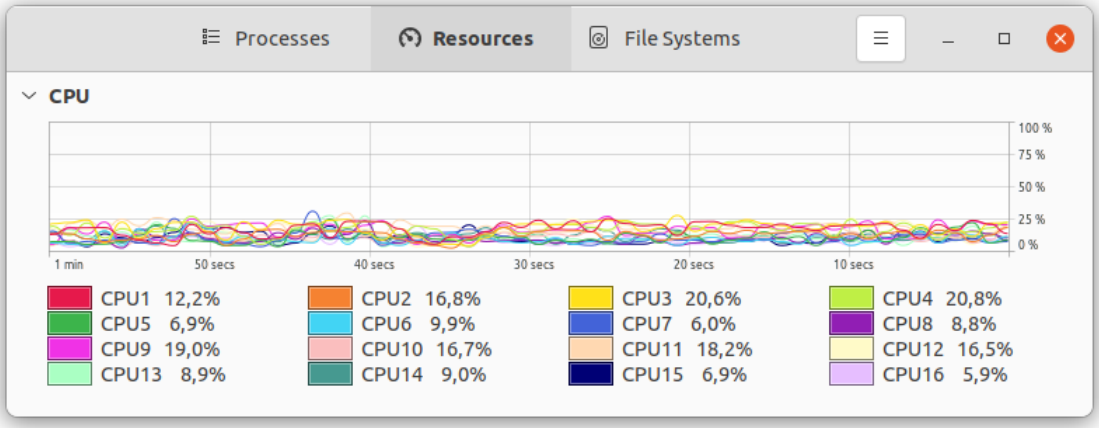

Figure A. 6. CPU load while processing "2013_04_04_GNSS_SIGNAL_at_CTTC_SPAIN.dat" signal.

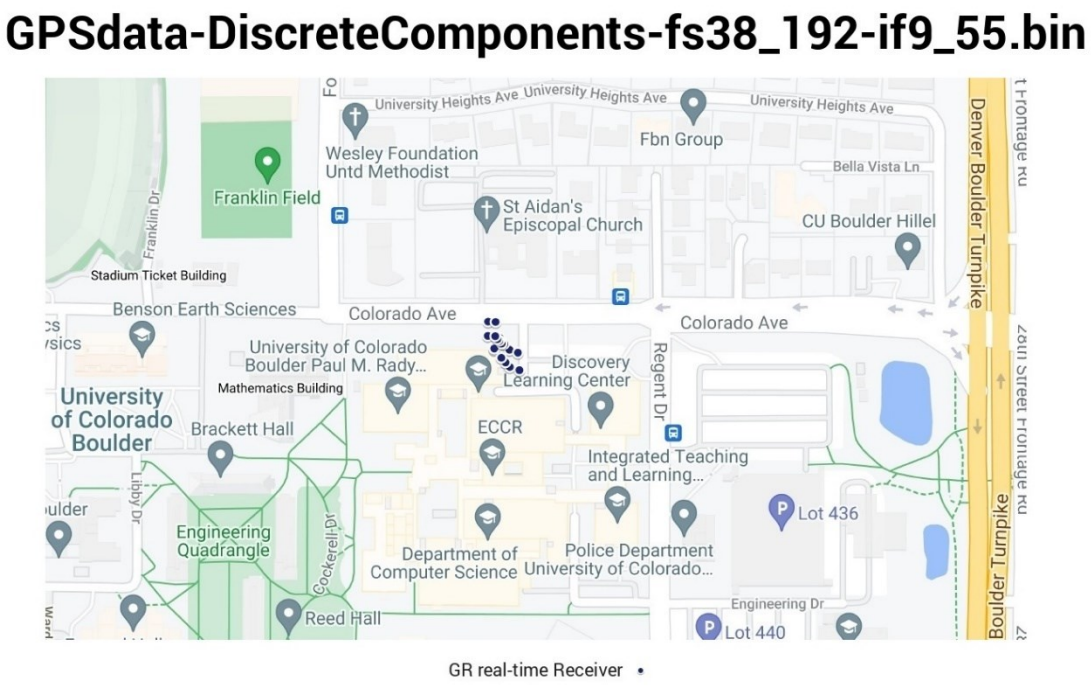

Figure A. 8. Results from flowgraph on Figure A. 1 execution, plotted in Google Maps

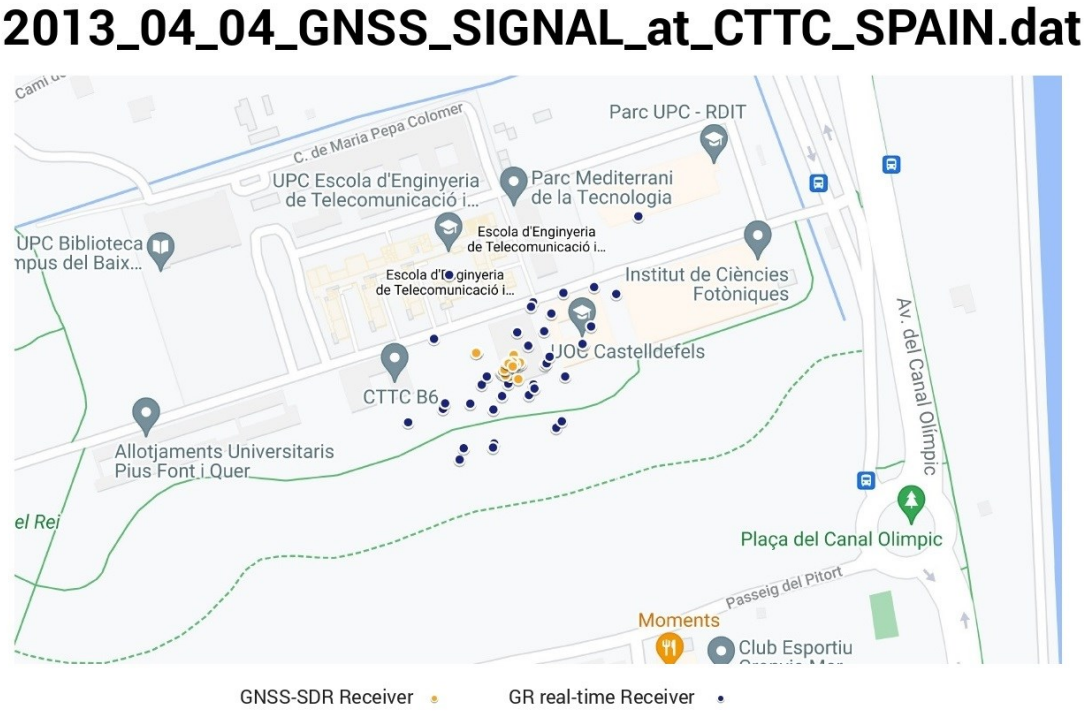

Figure A. 7. Results from flowgraph on Figure A. 2 execution, plotted in Google Maps.

BIBLIOGRAPHY

Akos, D., 1997. *A Software Radio Approach to GNSS Receiver Design,* Ohio: Ohio University.

Anon., 2021. *GRC Guided Tutorial.* [Online]
Available at: https://wiki.gnuradio.org/index.php/Guided_Tutorial_GRC
[Accessed 9 10 2021].

Arinc Research Corporation, 1991. *ICD-GPS-200.* Fountain Valley(California): Arinc Research Corporation.

Borre, K. et al., 2007. *A Software-Defined GPS and Galileo Receiver. A single-Frequency Approach.* ed. (): Springer.

Fernandez, C., Arribas, J. & Others, 2011. *GNSS SDR.* [Online]
Available at: https://gnss-sdr.org/
[Accessed 4 11 2021].

Gaël Guennebaud, B. J. a. o., 2010. *Eigen v3.* [Online]
Available at: http://eigen.tuxfamily.org
[Accessed 18 10 2021].

GNU Radio, 2016. *About.* [Online]
Available at: https://www.gnuradio.org/about/
[Accessed 11 10 2021].

RTKLIB, 2013. *RTKLIB.* [Online]
Available at: http://www.rtklib.com/
[Accessed 18 11 2021].

Schafer, R. W. & Oppenheim, A. V., . *Digital Signal Processing.* ed. (): Prentice Hall.

Seung-Hyun, Y. et al., 2008. *GPS Receiver with Enhanced User Positioning Time.* s.l., s.n.

Tsui, J. B.-y., 2000. *Fundamentals of Global Positioning System Receivers: A Software Approach.* New York, NY: John Wiley & Sons.

Wiki GNU Radio, 2018. *Gnu Radio Companion Guided Tutorial.* [Online]
Available at: https://wiki.gnuradio.org/index.php/Guided_Tutorial_GRC
[Accessed 11 10 2021].

Wiki GNU Radio, 2021. *Message Passing.* [Online]
Available at: https://wiki.gnuradio.org/index.php/Message_Passing
[Accessed 11 10 2021].

Wiki GNU Radio, 2021. *Out of tree modules.* [Online]
Available at: https://wiki.gnuradio.org/index.php/OutOfTreeModules
[Accessed 17 11 2021].

Wiki GNU Radio, 2021. *PMT Pairs.* [Online]
Available at: https://wiki.gnuradio.org/index.php/Polymorphic_Types_(PMTs)#Pairs
[Accessed 15 10 2021].

Wiki GNU Radio, 2021. *Polymorphic Types.* [Online]
Available at: https://wiki.gnuradio.org/index.php/Polymorphic_Types_(PMTs)
[Accessed 11 10 2021].

Ziemer, R. E. &. P. R. L., 1985. *Digital Communications and Spread Spectrum Systems.* New York: MacMillan.