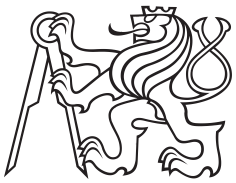
Bachelor Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Radioelectronics

Multiconstellation GNSS Receiver

Signal Tracking

Anastas Nikolov

Supervisor: Ing. Jiří Svatoň, PhD.
May 2022

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Nikolov Anastas** Personal ID number: **492112**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Radioelectronics**

Study program: **Open Electronic Systems**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Multiconstellation GNSS Receiver, Signal Tracking

Bachelor’s thesis title in Czech:

Multikonstelační GNSS přijímač, sledování signálu

Guidelines:

The thesis objective is a GNSS signal feedback tracking system analysis. The student will develop an SDR-like MATLAB simulation environment to simulate general GNSS signal processing at first. These simulations will be used to compare and debug the existing real HW multi-constellation GNSS receiver designed in the department. The work will come out of the traditional GPS L1 CA signal. However, the goal is to deploy the simulation environment to receive other signals like Galileo E1 (testing of data vs. pilot and BPSK-like vs. BOC approach) and the BPSK 10 class signals. These results will be potentially implemented to the HW receiver platform.

Bibliography / sources:

[1] Borre, K., Akos, D. M., Bertelsen, N., Rinder, P., Jensen, S. H., A software-defined GPS and Galileo receiver: a single-frequency approach, Springer Science & Business Media, 2007.
[2] Kaplan, E. D., Hegarty, C., Understanding GPS/GNSS: principles and applications, Third Edition, Artech house, 2019.
[3] Ziedan, N., Global Navigation Satellite System (GNSS) Receivers for Weak Signals, Artech house, 2006.
[4] Hrdina, Z., Pánek, P., Vejražka, F., Rádiové určování polohy: Družicový systém GPS, Praha: ČVUT, 1995, ISBN 80-01-01386-3.

Name and workplace of bachelor’s thesis supervisor:

Ing. Jiří Svatoň, Ph.D. Department of Radioelectronics FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **21.01.2022** Deadline for bachelor thesis submission: **20.05.2022**

Assignment valid until: **30.09.2023**

Ing. Jiří Svatoň, Ph.D.	doc. Ing. Stanislav Vítek, Ph.D.	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

. _____ Date of assignment receipt

_____ Student’s signature

Acknowledgements

I would like to express gratitude to my supervisor, Ing. Jiří Svatoň, PhD., for his guidance and expertise. His insight was invaluable for writing this thesis.
It was a great pleasure to work on this thesis and I hope that reader will find it both informative and enjoyable.

Declaration

I declare, that I have worked up the submitted work independently and that I have mentioned all the sources used in accordance with methodical instruction about adhering ethical principles when preparing university final works.
In Prague, 20. 5. 2022.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.
V Praze dne 20. 5. 2022

. . . . . .

Abstract

Signal tracking is a vital part of any GNSS receiver. It is a crucial part to remain synchronized with the incoming signal and to be able to demodulate transmitted data. However, many engineers approach feedback systems as a black box, and this thesis aims to describe *lock loops* used in GNSS receivers in detail.

To support this thesis, Software Defined Radio like simulator was developed to help analyze the feedback systems.

One of the more significant problems with GNSS lock loops is that they are tied together. When one loop does not work correctly, the whole system crumbles. A generator of authentic GNSS signals was developed and implemented as part of the SDR-like simulator to make isolated testing of individual loops possible. These signals have customizable properties such as permanent carrier/code wipe off, which allows us to test individual loops separately.

Towards the end of the thesis, implemented lock loops are compared with the theory of general feedback systems. It is illustrated that they behave according to the theoretical background even though it may not be apparent in some cases.

Finally, it is demonstrated that the developed SDR-like simulator is correctly tracking live GNSS signals recorded at CTU in Prague.

Keywords: GNSS, signal tracking,feedback loops, GPS, Galileo, DLL, FLL, PLL

Supervisor: Ing. Jiří Svatoň, PhD.

Abstrakt

Sledování signálu je důležitou součástí každého GNSS přijímače. Jde o klíčovou součást, která umožňuje zůstat v synchronizaci s příchozím signálem a demodulovat přenášená data. Mnoho inženýrů však přistupuje k zpětnovazebním systémům jako k černé skříňce a cílem této práce je tyto systémy používané v GNSS přijímačích podrobně popsat.

Souběžně s touto prací byl vyvinut simulátor podobný softwarově definovanému rádiu, který pomáhá analyzovat zpětnovazební systémy.

Jedním z významnějších problémů se zpětnovazebními smyčkami v GNSS přijímačích je jejich provázanost. Když jedna smyčka přestane správně fungovat, zhroutí se celý systém. Proto byl vyvinut generátor autentických GNSS signálů, který byl implementován do vyvíjeného simulátoru, aby bylo možné testovat jednotlivé smyčky izolovaně. Tyto signály mají nastavitelné vlastnosti, jako je trvalé odstranění nosné/kódu, což nám umožňuje testovat jednotlivé smyčky odděleně.

Ke konci práce jsou implementované smyčky porovnány s teorií obecných zpětnovazebních systémů. Je ukázáno, že se chovají v souladu s teorií, přestože to v některých případech nemusí být zřejmé. Na závěr je demonstrováno, že vyvinutý simulátor podobný SDR správně sleduje živé signály GNSS zaznamenané na ČVUT v Praze.

Klíčová slova: GNSS, seldování signálu, zpětnovazební smyčky, GPS, Galielo, DLL, FLL, PLL

Překlad názvu: Multikonstelační GNSS přijímač — sledování signálu

Contents

Figures

Tables

Chapter 1

Satellite Navigation

Most navigational systems are made of a beacon and onboard equipment. In satellite navigation, the satellite is the beacon [1].

1.1 Doppler navigation systems

Doppler navigation systems are based on the Doppler effect due to the movement of satellites, transmitting at a constant frequency. If satellite A is transmitting at frequency f_0, then the signal on the receiver end will be of frequency $f_d \neq f_0$. The received signal is input into a frequency mixer together with a signal from a local oscillator of frequency f_0. Output signal of the frequency mixer is of frequency $f_0 - f_d$.
Together with time-stamps that are contained in the signal, it is possible to determine the position or speed of the user and/or time synchronization.

1.2 Code systems

Code systems are the most widely used. Time-stamps and ephemerides (position of the satellite in 3D-space) are extracted from the signal.
There are two types of code systems - active and passive.

1.2.1 Active code systems

The user needs an active radio because active communication with a reference station is needed. These are typical request-response systems. The reference station sends a request with a user ID. Satellites serve as a middleman - they get the user's request and the station's response. The station evaluates delays of responses from multiple satellites and calculates the position of the user (positions of satellites are known from ephemerides, so the only unknown is the position of the user).
Active code systems have disadvantages too. For example, they are not very secure (the user has to be active), and the system can be overloaded [1].

1.2.2 Passive code systems

The user calculates the distance from the satellite from the time that has passed between signal transmission and reception. With this information together with ephemerides, the user can determine his position.
The problem is that the time base of the user is shifted by an unknown time interval. This interval Δt can be transformed to a distance $b = c\Delta t$ [1]. This has quite an unpleasant consequence - instead of three equations for three unknowns, we now have to deal with four.
Usually, we approach this by synchronizing the satellite signal with a local copy. When in sync, we know the delay τ_m between the start of the received and local sequence. From τ_m, *pseudorange* is calculated. To solve our system of equations, we need at least four satellites.

Chapter 2

Radio Transmission

This chapter introduces basics of radio communication theory.

2.1 Complex Signal Reconstruction

Since only *real* signals can propagate through space, it might not be clear if it is even possible to use complex signals for radio communication. Assume a complex signal of form $s(t) = a + jb \mid a, b \in \mathbb{R}$, that we want to transmit. We multiply this signal by carrier $e^{-2\pi j f_0 t}$, where f_0 is the carrier frequency. Using Euler formula we can write it as $\cos(2\pi f_0 t) - j\sin(2\pi f_0 t)$. As stated, only *real* signals can be transmitted, so let's just do that.

$$\begin{aligned} &\mathbf{Re}\left[(a+jb)\cdot\left(e^{-2\pi j f_0 t}\right)\right] = \\ &= \mathbf{Re}\left[(a+jb)\cdot(\cos(2\pi f_0 t) - j\sin(2\pi f_0 t))\right] = \\ &= a\cos(2\pi f_0 t) + b\sin(2\pi f_0 t) \end{aligned} \tag{2.1}$$

The imaginary part is not lost and can be retrieved by the receiver. Ideally, the signal is transmitted without any additional noise and without the Doppler effect (or we can also say that the Doppler effect is known and incorporated in f_0). We split the received signal into two branches - In-phase (I) and Quadrature (Q).

$$\begin{aligned} &I(t) = (a\cos(2\pi f_0 t) + b\sin(2\pi f_0 t))\cdot\cos(2\pi f_0 t) = \\ &= a\cos^2(2\pi f_0 t) + b\sin(2\pi f_0 t)\cos(2\pi f_0 t) = \\ &= \frac{a}{2}(1+\cos(4\pi f_0 t)) + \frac{b}{2}\sin(4\pi f_0 t) \xrightarrow{Lowpass} \frac{1}{2}a = \frac{1}{2}\mathbf{Re}[s(t)] \end{aligned} \tag{2.2}$$

$$\begin{aligned} &Q(t) = (a\cos(2\pi f_0 t) + b\sin(2\pi f_0 t))\cdot(\sin(2\pi f_0 t)) = \\ &= a\cos(2\pi f_0 t)\sin(2\pi f_0 t) + b\sin^2(2\pi f_0 t) = \\ &= \frac{a}{2}\sin(4\pi f_0 t) + \frac{b}{2}(1-\cos(4\pi f_0 t)) \xrightarrow{Lowpass} \frac{1}{2}b = \frac{1}{2}\mathbf{Im}[s(t)] \end{aligned} \tag{2.3}$$

By combining the two branches we get

$$s(t) = \frac{1}{2}\left(I(t) + jQ(t)\right) \tag{2.4}$$

2.1.1 Simulation

We can use Simulink to demonstrate this. In the figure below is our system model that does all of the calculations mentioned above, $s(t) = 5 + j$.

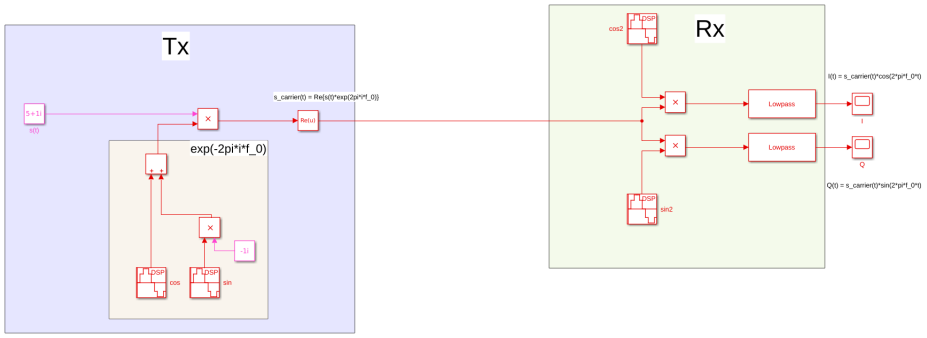


Figure 2.1: Transmitter-Receiver of a complex signal

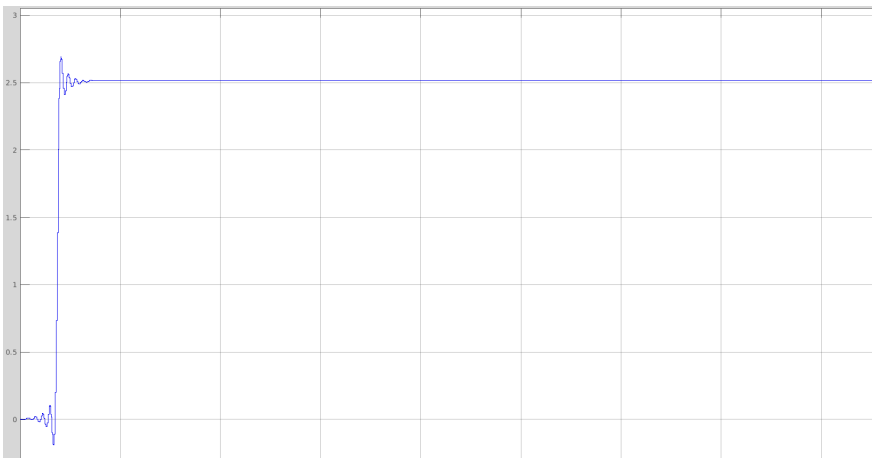


Figure 2.2: In-phase branch of the signal

Figure 2.3: Quadrature branch of the signal

2.1.2 Doppler effect

Let us now consider the same signal, but the transmitter is moving, causing a Doppler shift that is unknown to the receiver. We shall denote the frequency shift caused by the Doppler effect f_d. For the sake of compactness we will introduce $\Delta_f = f_0 + f_d$. There is also an unknown phase shift φ. Nothing changes on the transmitter-end (apart from the shift in frequency and phase), leaving us with:

$$s(t) = a\cos\left(2\pi\Delta_f t + \varphi\right) + b\sin\left(2\pi\Delta_f t + \varphi\right) \tag{2.5}$$

Neither the frequency nor the phase shift is known to the receiver, meaning that we will be multiplying the I and Q branches by carriers of frequency f_0.

$$\begin{aligned}
&I\left(t\right) = \left(a\cos\left(2\pi\Delta_f t + \varphi\right) + b\sin\left(2\pi\Delta_f t + \varphi\right)\right)\cdot\cos\left(2\pi f_0 t\right) = \\
&= a\left(\frac{e^{2\pi j\Delta_f t + j\varphi} + e^{-2\pi j\Delta_f t - j\varphi}}{2}\cdot\frac{e^{2\pi j f_0 t} + e^{-2\pi j f_0 t}}{2}\right) + \\
&+ b\left(\frac{e^{2\pi j\Delta_f t + j\varphi} - e^{-2\pi j\Delta_f t - j\varphi}}{2j}\cdot\frac{e^{2\pi j f_0 t} + e^{-2\pi j f_0 t}}{2}\right) \xrightarrow{Lowpass} \\
&a\frac{e^{2\pi j f_d t + j\varphi} + e^{-2\pi j f_d t - j\varphi}}{4} + b\frac{e^{2\pi j f_d t + j\varphi} - e^{-2\pi j f_d t - j\varphi}}{4j} = \\
&= \frac{a}{2}\cos\left(2\pi f_d t + \varphi\right) + \frac{b}{2}\sin\left(2\pi f_d t + \varphi\right)
\end{aligned} \tag{2.6}$$

$$
\begin{aligned}
&Q(t) = (a\cos(2\pi\Delta_f t + \varphi) + b\sin(2\pi\Delta_f t + \varphi)) \cdot \sin(2\pi f_0 t) = \\
&= a\left(\frac{e^{2\pi j\Delta_f t + j\varphi} + e^{-2\pi j\Delta_f t - j\varphi}}{2} \cdot \frac{e^{2\pi j f_0 t} - e^{-2\pi j f_0 t}}{2j}\right) + \\
&+ b\left(\frac{e^{2\pi j\Delta_f t + j\varphi} - e^{-2\pi j\Delta_f t - j\varphi}}{2j} \cdot \frac{e^{2\pi j f_0 t} - e^{-2\pi j f_0 t}}{2j}\right) \xrightarrow{Lowpass} \\
&a\frac{-e^{2\pi j f_d t + j\varphi} + e^{-2\pi j f_d t - j\varphi}}{4j} + b\frac{e^{2\pi j f_d t + j\varphi} + e^{-2\pi j f_d t - j\varphi}}{4} = \\
&= \frac{b}{2}\cos(2\pi f_d t + \varphi) - \frac{a}{2}\sin(2\pi f_d t + \varphi)
\end{aligned}
\tag{2.7}
$$

With unknown frequency/phase shift, we cannot reconstruct the signal. In the figures below, we once again simulated our system. Here $f_0 = 20Hz$ and $f_d = 5Hz$.

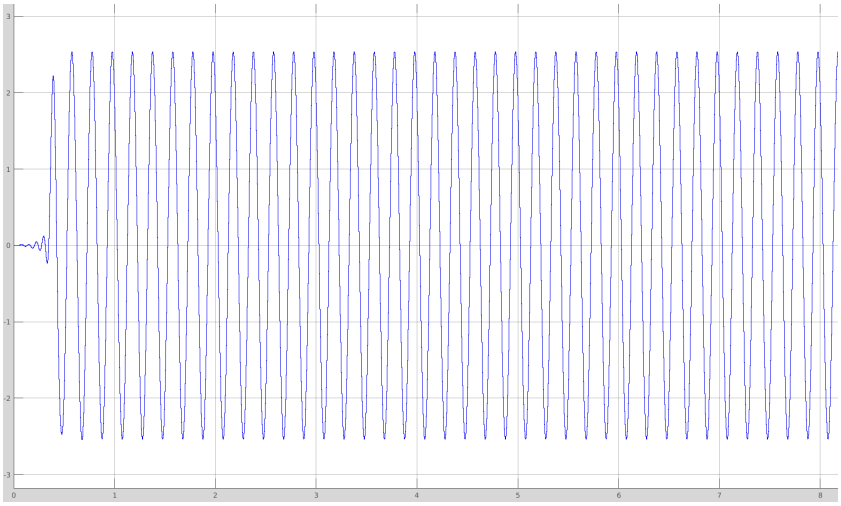


Figure 2.4: In-phase branch, Doppler effect

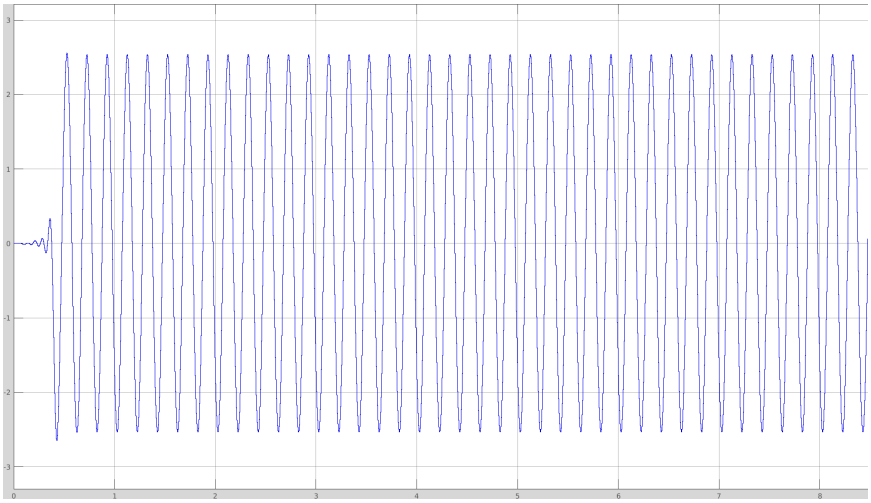


Figure 2.5: Quadrature branch, Doppler effect

2.2 GNSS Signal Structure

There are two main requirements for GNSS signals - precise measurement of the exact time the signal was received and transmission of navigation data. Typically, Binary Phase Shift Keying (BPSK) modulation is used, but some modern GNSS systems use Binary Offset Carrier (BOC), which has some pleasant properties that will be discussed later.
Modern GNSS systems use pilot signals. The total power of one signal is split into two components - data and pilot. Since the pilot component is not modulated by navigation data, it provides easier signal tracking.
Many of the modern GNSS systems utilize secondary codes, a periodic binary sequence that reduces interference between individual GNSS systems. Sometimes, secondary codes are called synchronization codes as they are used for bit synchronization in GNSS receivers.

2.2.1 Binary Phase Shift Keying

Generally, a signal formed by linear modulation can be mathematically described by the following formula.

$$s(t) = \sum_{n} q_n g(t - nT_q) \tag{2.8}$$

Modulation is uniquely defined by symbol pulse $g(t)$ and channel symbols q_n, for BPSK $q_n \in \{\pm 1\}$, T_q is the symbol period. In radio electronics, we commonly work with *complex envelope* of the signal, which for the small price of losing information about the carrier frequency, simplifies mathematical

operations. For data and pilot components, we arrive at

$$\begin{aligned} \tilde{s}_d(t) &= 2\sqrt{P}d(t)prn(t) \\ \tilde{s}_p(t) &= 2\sqrt{P}prn(t) \end{aligned} \tag{2.9}$$

where P is the power of the signal, *d(t)* is signal with navigational data and *prn(t)* is pseudo-random noise.

$$\begin{aligned} d(t) &= \sum_{n=-\infty}^{\infty} q_n g(t - nT_q) \\ prn(t) &= \sum_{n=-\infty}^{\infty} q'_n g(t - nT'_q) \end{aligned} \tag{2.10}$$

Here, modulation pulse g is REC pulse with period T_q, respectively T'_q. Nominal chip frequency is then $f_{\text{chipn}} = 1/T'_q$ (eg. 1.023 MHz for GPS L1 and Galileo E1C).
Few more steps are required to acquire the signal that can be modulated onto the carrier. First, the data sequence (q_n) has to be re-sampled (let us denote it as $\hat{q}_n$) to match the speed of q'_n. These are then multiplied, creating new sequence $\xi_n = \hat{q}_n \cdot q'_n$.

$$\xi(t) = \sum_{n=-\infty}^{\infty} \xi_n g(t - nT'_q) \tag{2.11}$$

The carrier wave can be modulated with this signal, the complex envelope then takes form

$$\tilde{s}_d(t) = 2\sqrt{P}\xi(t) \tag{2.12}$$

2.2.2 Binary Offset Carrier

To reduce interference with other signals, modern GNSS systems (such as Galileo) are using BOC modulation, which utilizes the so-called *digital subcarrier* to concentrate signal power within specific parts of the band. Two variants of BOC modulation are used, and their complex envelopes look like this.

$$\begin{aligned} \tilde{s}_{\sin}(t) &= 2\sqrt{P}d(t)prn(t)\text{sign}\left(\sin\left(\frac{2\pi}{2T_{sc}}t\right)\right) \\ \tilde{s}_{\cos}(t) &= 2\sqrt{P}d(t)prn(t)\text{sign}\left(\cos\left(\frac{2\pi}{2T_{sc}}t\right)\right) \end{aligned} \tag{2.13}$$

Where $T_{sc} = \frac{1}{2f_s}$ and f_s is the sub-carrier frequency.

2.2.3 Code Division Multiple Access

The GNSS satellites share the same communication channel at the same time. Some multiple access technique is needed to distinguish between individual signals. While GLONASS uses frequency division multiple access (FDMA), GPS and Galileo are using a technique that allows the use of common carrier frequency. This is possible by assigning each satellite a unique pseudo-random noise (PRN) code. These have low cross-correlation properties (and high auto-correlation properties) with PRN codes of other satellites. Now it is clear why this technique is called code division multiple access (CDMA). Due to mentioned correlation properties of PRN codes, we can recover the original signal with little interference from other signals.

2.3 GNSS Receiver

The signal processing for satellite navigation systems is based on a channelized structure. This is true for both GPS and Galileo [3, p. 87].
Before the receiver can allocate channels to individual satellites, it must first determine which satellites are currently visible. This is done by signal acquisition (see chapter 3).
When a satellite is acquired, a digital channel is assigned to track it. The channel can be represented with the following block diagrams.

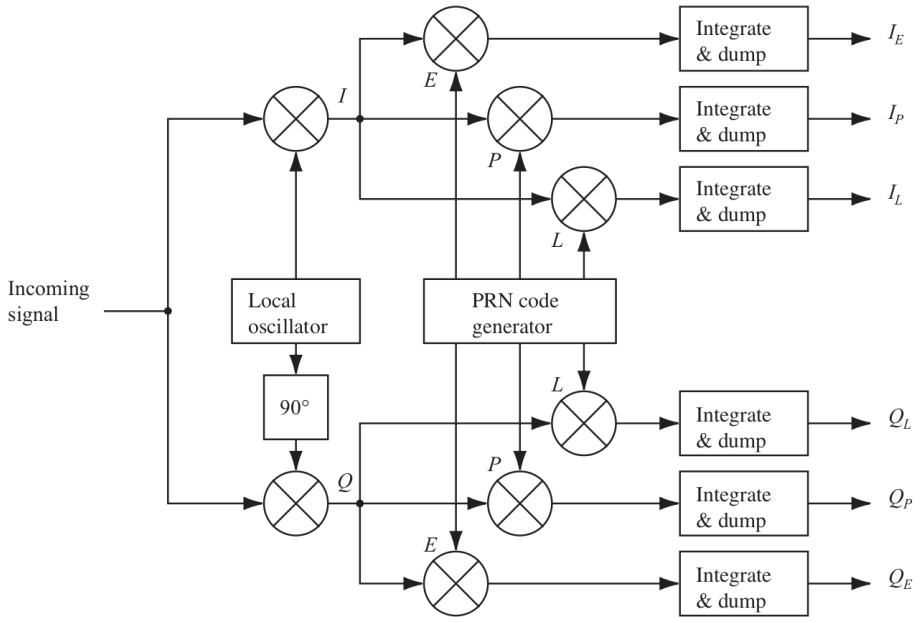


Figure 2.6: Digital channel block diagram [3, p. 87]

The incoming signal is first multiplied by a locally generated carrier wave and its $\pi/2$ shifted version to split the signal into in-phase and quadrature branches. These are multiplied by three versions of the local PRN code replica

- each with some phase offset. The resulting signals are integrated and output for further processing in discriminators.

Chapter 3

GNSS Signal Acquisition

The purpose of the acquisition is to determine visible satellites and coarse values of carrier frequency and code phase of the satellite signals [3, p. 92].

3.1 Concept

The signal acquisition is the primary estimate of replica alignment before the transition to a stable state of tracking of the alignment by feedback loops. These work correctly only in a certain pull-in range, and the acquisition provides a starting point in this range that can be further refined.
The satellites are distinguishable by 32 unique PRN sequences. To generate a PRN replica perfectly aligned with the incoming code, it is necessary to find the code phase. Then the incoming code can be removed from the signal. For this purpose, the PRN is designed to have high correlation when aligned perfectly and almost no correlation, even for small lag. The third parameter is the carrier frequency. The "clean" carrier frequency is predetermined, but there is always some deviation due to the Doppler effect. The frequency can deviate up to ±10 kHz [3, p. 92]. The acquisition algorithm provides only a coarse estimate of this deviation. Still, the error should not be higher than 500 Hz.

3.2 Serial Search Algorithm

This is the least hardware-demanding method as it requires a single correlator. However, it has to go through all of the parameters serially.
This algorithm has to search through all possible frequencies (as stated, the worst case is ±10 kHz) in steps of some $\Delta_{doppler}$ (this can be the usually tolerated 500 Hz). It also has to search through all possible code phases (1023 possibilities for each frequency step. If we use our example numeric values, this results in 41 943 combinations that the algorithm has to evaluate. The likelihood function can be written as

$$\mathcal{R}(\tau, f_d) = \int_{\tau=0}^{T_{\text{code}}} x(t) prn(t-\tau) e^{2\pi j f_d t} dt \tag{3.1}$$

where $x(t)$ is the observed signal and prn is the local replica of PRN code. The estimated parameters are then

$$(\hat{\tau}, \hat{f}_d) = \underset{\tau, f_d}{\operatorname{argmax}} |\mathcal{R}(\tau, f_d)| \tag{3.2}$$

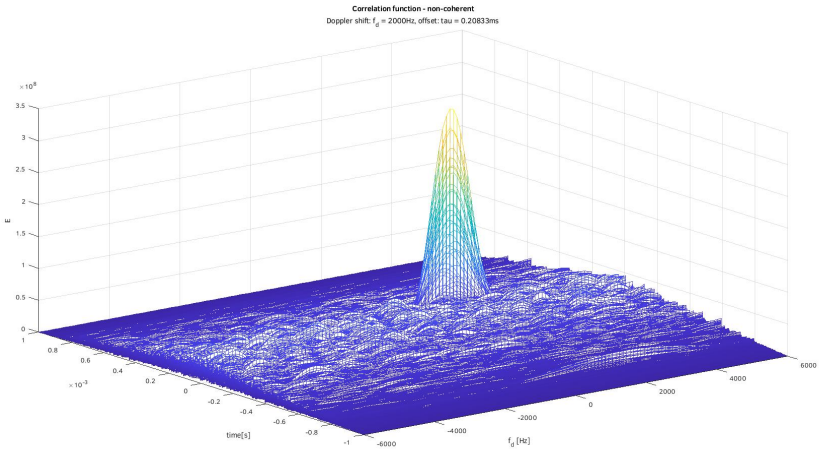


Figure 3.1: Cross Ambiguity Function

3.3 Parallel in Code Phase Search Algorithm

The parallel search algorithm operates in the frequency domain. This is possible due to the Wiener-Kinchin theorem, which defines a connection between spectral and correlation characteristics of the signal. The algorithm computes a circular cross correlation of observed signal and local replica of PRN. The circular cross correlation is done in frequency domain usign fast Fourier transform (FFT). The block diagram (fig. 3.2) of parallel in code phase search algorithm can be mathematically expressed with equation 3.3. The output of acquisition script (estimated parameters) are found using equation 3.4.

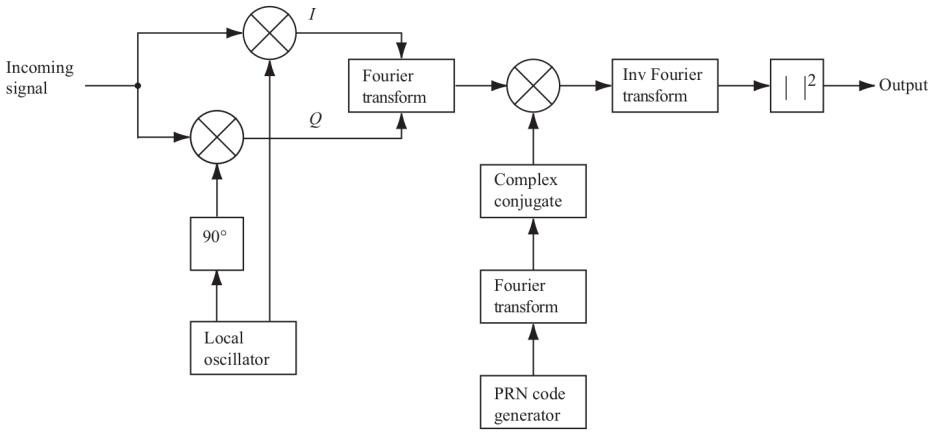


Figure 3.2: Parallel in code phase search algorithm block diagram [3, p. 83]

$$\mathcal{R}(t, f_d) = \left|\text{IFFT}\{\text{FTT}\{x(t)\} \cdot \overline{\text{FFT}\{prn(t)e^{2\pi j f_d t}\}}\}\right|^2 \tag{3.3}$$

$$(\hat{\tau}, \hat{f}_d) = \underset{\tau, f_d}{\text{argmax}} |\mathcal{R}(\tau, f_d)| \tag{3.4}$$

This method parallelises the search for the code phase (we don't need to step through 1023 different code phases as in serial search). The complexity of FFT increases logarithmically with a growing number of samples, which is why this method is very popular.

Chapter 4

Lock Loop Feedback Tracking

4.1 Carrier Tracking

We use two feedback loops to track the carrier wave signal - phase lock loop (PLL) and frequency lock loop (FLL). While secondary codes in pilot signal components of new signals like E1C are known sequences, in standard signals like GPS L1 with data or data channel like E1B we do not have a priori knowledge of navigation data modulated on the primary codes. Thus ordinary PLL is out of the question as it is sensitive to 180° phase shifts that occur every navigation bit transition. For this reason, Costas loops (which are insensitive to data modulation, see figure 7.23) are used.

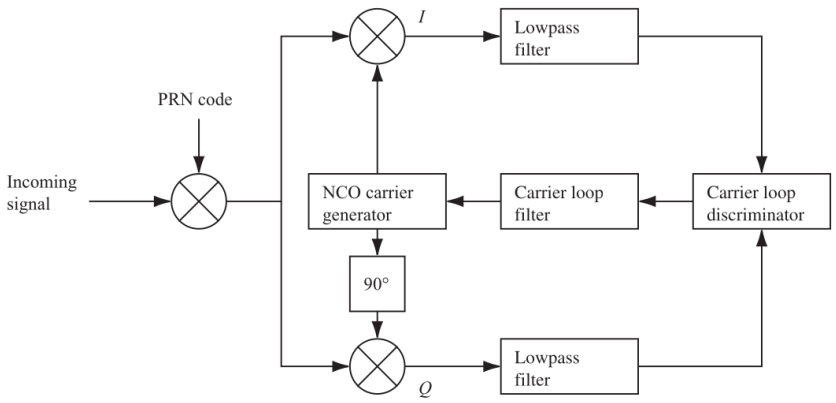


Figure 4.1: Costas loop used to track the carrier wave [3, p. 111]

4.1.1 Phase Lock Loop Discriminator

PLL discriminator outputs phase error. The table below compares the error outputs of four different PLL discriminators. In an operational environment, the PLL discriminator error signals are indeed periodic [2, p. 460].

Discriminator Algorithm	*Output Phase Error*	*Characteristics*
$ATAN2(Q_P, I_P)$	ϕ	Four-quadrant arctangent. Optimal (maximum likelihood estimator) at high and low SNR. Slope not signal amplitude dependent. High computational burden. Usually a table look-up implementation.
$\frac{Q_P}{Ave\sqrt{I_P^2 + Q_P^2}}$	$\sin\phi$	Q_P normalized by averaged prompt envelope. Slightly outperforms four-quadrant arctangent. Q_{PS} approximates ϕ to $\pm 45°$. Normalization provides insensitivity at high and low SNR. Also keeps slope not signal amplitude dependent. Low computational burden.

Table 4.1: Ordinary PLL Discriminators [2, p. 460]

Discriminator Algorithm	*Output Phase Error*	*Characteristics*
$Q_P \times I_P$	$\sin 2\phi$	Classic Costas analog discriminator. Near optimal at low SNR. Slope proportional to signal amplitude squared A^2. Moderate computational burden.
$Q_P \times \text{Sign}\ (I_P)$	$\sin\phi$	Decision directed Costas. Near optimal at high SNR. Slope proportional to signal amplitude A. Least computational burden.
Q_P/I_P	$\tan\phi$	Suboptimal but good at high and low SNR. Slope not signal amplitude dependent. Higher computational burden. Divide by zero error at $\pm 90°$.
ATAN (Q_P/I_P)	ϕ	Two-quadrant arctangent. Optimal (maximum likelihood estimator) at high and low SNR. Slope not signal amplitude dependent. Highest computational burden. Usually a table look-up implementation.

Table 4.2: Costas Loop Discriminators [2, p. 461]

4.1.2 Frequency Lock Loop Discriminator

FLL discriminator outputs frequency error. The table below compares common FLL discriminator algorithms. In an operational environment, the FLL discriminator error signals are also periodic. However, their amplitudes are severely attenuated beyond the frequency limits of their pull-in ranges by the narrow bandwidths of their FLL tracking loops. In the presence of noise in the prompt I and Q signals, the slopes of all of the FLL discriminator outputs tend to flatten as the noise levels increase. Thus, they are linear only near the 0-Hz error region [2, p. 464].

Discriminator Algorithm	*Output Frequency Error Characteristics*	
$\frac{cross}{(t_2 - t_1)}$	$\frac{\sin(\phi_2 - \phi_1)}{t_2 - t_1}$	Near optimal at low SNR. Slope proportional to signal amplitude squared A^2. Least computational burden.
where: $cross = I_{P1} \times Q_{P2} - I_{P2} \times Q_{P1}$		
$\frac{(cross) \times sign(dot)}{(t_2 - t_1)}$	$\frac{\sin[2(t_2 - t_1)]}{t_2 - t_1}$	Decision directed. Near optimal at high SNR. Slope proportional to signal amplitude A. Moderate computational burden.
where: $dot = I_{P1} \times I_{P2} + Q_{P1} \times Q_{P2}$		
$\frac{ATAN2(cross, dot)}{(t_2 - t_1)}$	$\frac{\phi_2 - \phi_1}{t_2 - t_1}$	Four-quadrant arctangent. Maximum likelihood estimator. Optimal at high and low SNR. Slope not signal amplitude dependent. Highest computational burden. Usually a table look-up implementation.

Table 4.3: Common FLL Discriminators [2, p. 462]

4.2 Code Tracking

The goal of a code tracking loop is to keep track of the code phase of a specific code in the signal [3, p. 113].
The code tracking loop used in GNSS receivers is called the early-late delay lock loop (DLL). This loop correlates the received signal with three replicas generated (usually) with $\pm\frac{1}{2}$ chip spacing, although, for BOC modulation, it is much narrower. These values are then compared to determine which correlates the most. We can increase or decrease the current code phase accordingly with this information.

4.2.1 Delay Lock Loop Discriminator

There are four code loop discriminators with different characteristics that might interest us. These are called delay lock loop (DLL) discriminators. All of these use the early (E) and late (L) correlator phases, and the coherent version also uses a prompt (P) signal. The coherent DLL provides superior performance when the carrier loop is in PLL [2, p. 465]. Under these circumstances, the signal and noise are in the I branch, and there is mostly noise in the Q branch.
The table 4.4 shows these four discriminators and their characteristics. All of the DLL discriminators can be normalized. Normalizing them removes sensitivity to signal amplitude fluctuations which boost performance under rapidly changing C/N_0. In consequence, the normalized DLL is independent of automatic gain control performance.

Discriminator Algorithm	*Characteristics*
$\frac{1}{2}\frac{E-L}{E+L}$ where $E=\sqrt{I_E^2+Q_E^2}, L=\sqrt{I_L^2+Q_L^2}$	Noncoherent early minus late envelope normalized by $E+L$ to remove amplitude sensitivity. High computational load. For 1-chip BPSK E-L correlator spacing, produces true tracking error within ±½-chip of input error (in the absence of noise). Becomes unstable (divide by zero) at ±1.5-chip input error, but this is well beyond code tracking threshold in the presence of noise.
$\frac{1}{2}\left(E^2-L^2\right)$	Noncoherent early minus late power. Moderate computational load. For 1-chip BPSK, E-L correlator spacing produces essentially the same error performance as ½ (E-L) envelope within ±½-chip of input error (in the absence of noise). Can be normalized with E^2+L^2.
$\frac{1}{2}\left[(I_E-I_L)I_P+(Q_E-Q_L)Q_P\right]$ (dot product) $\frac{1}{4}\left[(I_E-I_L)/I_P+(Q_E-Q_L)/Q_P\right]$ $\left(\text{normalized with } I_P^2 \text{ and } Q_P^2\right)$	Quasi-coherent dot product power. Uses all three correlators. Low computational load. For 1-chip BPSK E-L correlator spacing, it produces nearly true error output within ±½-chip of input (in the absence of noise). Normalized version shown second using I_P^2 and Q_P^2, respectively.
$\frac{1}{2}(I_E-I_L)I_P$ (dot product) $\frac{1}{4}\frac{(I_E-I_L)}{I_P}\left(\text{normalized with } I_P^2\right)$	Coherent dot product. Can be used only when carrier loop is in phase lock. Low computational load. Most accurate code measurements. Normalized version shown second using I_P^2.

Table 4.4: GNSS code loop discriminators and their characteristics [2, p. 466]

The normalized early minus late envelope discriminator is very popular because its noise-free output error is linear over a ±1-chip range and has a pull-in range of almost ±1.5-chip. However, the dot product power discriminator slightly outperforms it [2, p. 467].
Narrow early to late correlator separations are used to reduce multi-path error and measurement noise. The cost is reduced code tracking loop dynamic stress tolerance. If we use carrier-aided code tracking, then the carrier loop removes most of the code loop dynamic stress, thus enabling narrow correlator spacing, resulting in smaller code loop filter noise bandwidths after reaching a steady state.
The figure below visualizes how the early, prompt and late envelope amplitudes change as the phase of the replica code progresses compared to the received BPSK signal. For our purposes, the signal is without noise.

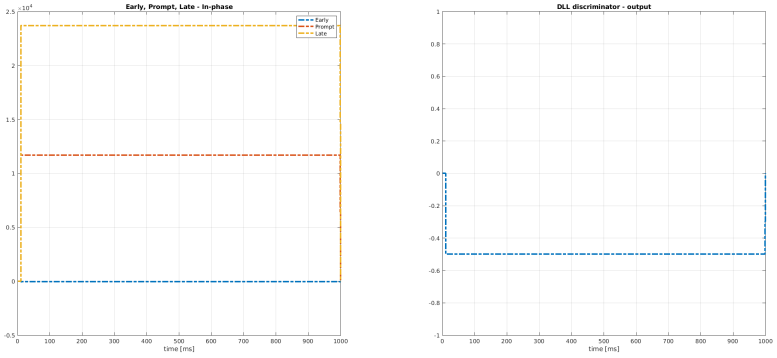


Figure 4.2: Replica 1/2 chip late

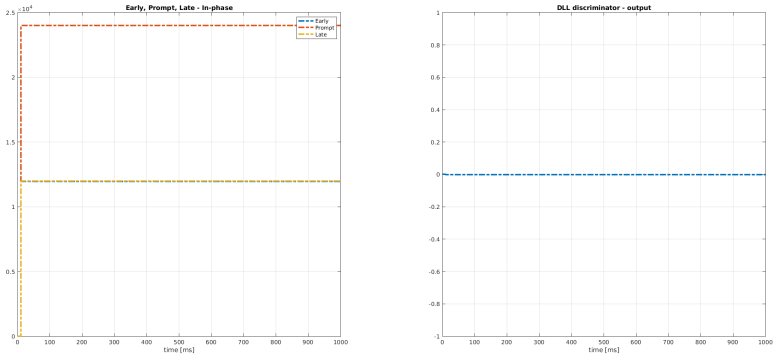


Figure 4.3: Replica aligned

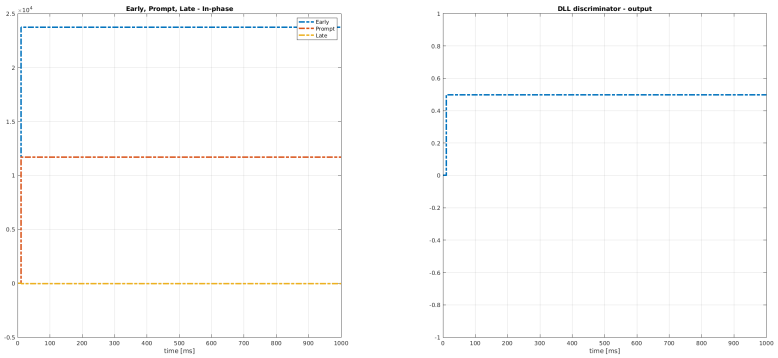


Figure 4.4: Replica 1/2 chip early

When the replica is aligned perfectly with the incoming signal (fig. 4.3), then $E = L$, and there is no DLL error. In case the replica and received signal are out of phase (fig. 4.2 and 4.4), then $E \neq L$ and envelopes are unequal *proportionally* to the code phase error between the replica and the signal. From the difference in amplitudes of the early and late envelopes, the code loop discriminator can determine the amount of error and even the direction (early or late). The error is filtered and applied to the NCO (output frequency is increased/decreased to correct the replica code generator phase).

4.3 Loop Filters

The objective of the loop filter is to reduce noise, thus producing an accurate estimate of the original signal. Loop filter order and noise bandwidth (B_n) determine the loop filter's response to signal dynamics. The order and noise bandwidth are determined based on the expected environment, receiver component noise contributions and desired precision. The loop filter is part of the feedback loop, meaning there are stability issues associated with the loop order. The Loop filter's output signal is subtracted from the original signal producing an error signal that is filtered and used to correct carrier and code replica signals.

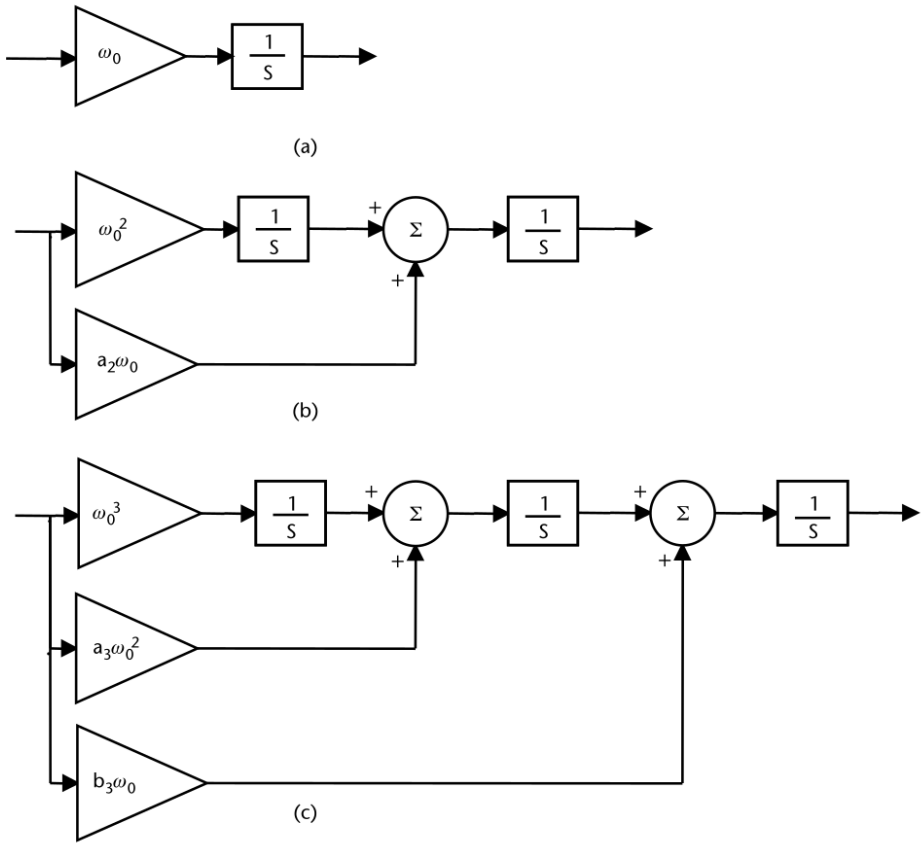


Figure 4.5: Block diagrams of: (a) first-, (b) second-, and (c) third-order analog loop filters [2, p. 473]

Loop Order	*Noise Bandwidth B_n (Hz)*	*Typical Filter Values*	*Steady State Error*	*Characteristics*
First	$\frac{\omega_0}{4}$	ω_0 $B_n = 0.25\omega_0$	$\frac{(dR/dt)}{\omega_0}$	Sensitive to velocity stress. Used in aided code loops.
Second	$\frac{\omega_0\left(1+a_2^2\right)}{4a_2}$	ω_0^2 $a_2\omega_0 = 1.414\omega_0$ $B_n = 0.53\omega_0$	$\frac{d^2R/dt^2}{\omega_0^2}$	Sensitive to acceleration stress. Used in aided code loops and aided and unaided carrier loops. Optimum damping factor $\delta = 0.707 = a_2/2$.
Third	$\frac{\omega_0\left(a_3b_3^2 + a_3^2 - b_3\right)}{4\left(a_3b_3 - 1\right)}$	ω_0^3 $a_3\omega_0^2 = 1.1\omega_0^2$ $b_3\omega_0 = 2.4\omega_0$ $B_n = 0.7845\omega_0$	$\frac{\left(d^3R/dt^3\right)}{\omega_o^3}$	Sensitive to jerk stress. Used in unaided carrier loops. Parameters provide fastest response to step function with minimal initial overshoot.

Table 4.5: Loop Filter Characteristics [2, p. 474]

4.3.1 PLL Filter

The PLL filter is typically second-order for moderate dynamic applications or third for higher dynamics. If second-order is used, it has one digital bilinear transform integrator and NCO. In the case of the third order, two digital bilinear transform integrators are used in combination with NCO.

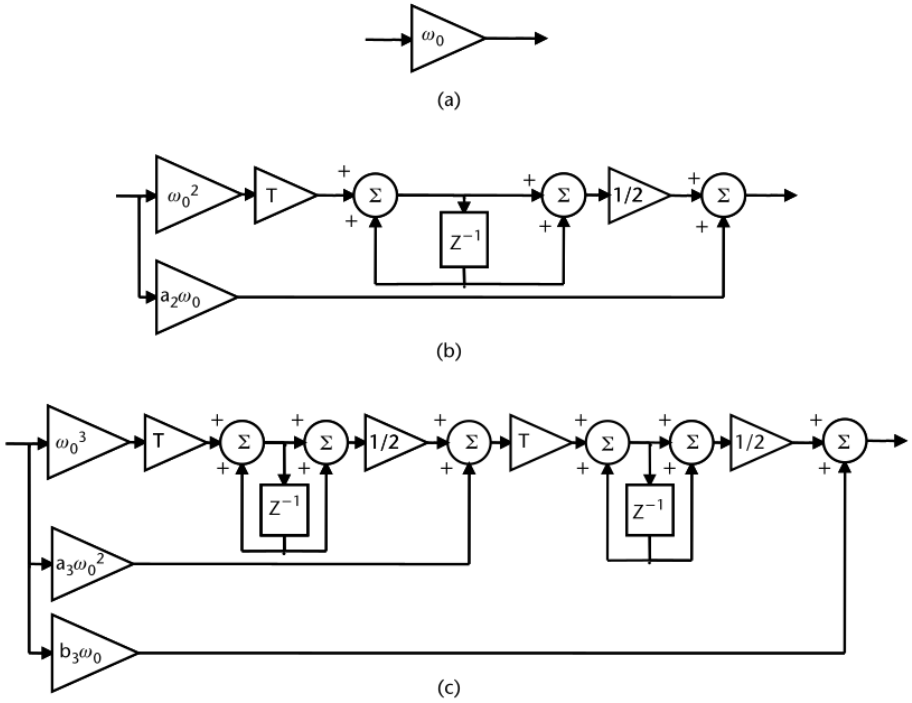


Figure 4.6: Block diagrams of (a) first, (b) second, and (c) third-order digital loop filters excluding last integrator (the NCO) [2, p. 475]

Costas PLL suffers squaring loss. This can only be reduced by increasing T [2, p. 476]. However, modern GNSS signals have pilot channels that operate with pure PLL discriminator, theoretically having zero squaring loss.

4.3.2 FLL Filter

FLL filter is typically one order lower than PLL but requires one more integrator. This is caused by FLL producing frequency error, but NCO fixes only phase error. Thus, the first order FLL has one digital bilinear transform integrator and NCO, and the second order FLL has two digital bilinear transform integrators and NCO.

4.3.3 FLL-assisted-PLL Filter

GNSS receivers that only rely on FLL are not as precise as receivers with PLL, leading to higher bit error rates in data demodulation. GNSS receivers that do not support FLL are vulnerable to sudden high dynamic stress. If we

have limited resources and still want to have the benefits of both FLL and PLL, so-called FLL-assisted-PLL may be the solution.

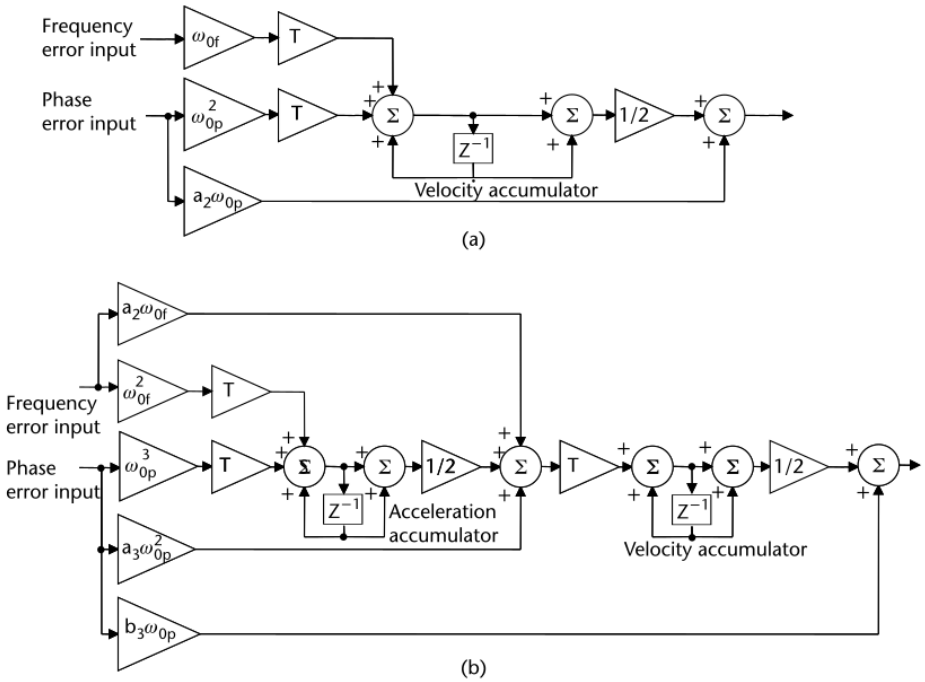


Figure 4.7: Block diagrams of FLL assisted PLL filters: (a) second-order PLL with first-order FLL as- sist, and (b) third-order PLL with second-order FLL assist [2, p. 477]

If the PLL error input is zeroed, the filter becomes pure FLL. If the FLL error is zeroed, it becomes pure PLL.
Here follows the loop closure process:

- Close in pure FLL
- Apply error inputs from both discriminators as an FLL assisted PLL until phase lock
- Convert to pure PLL until phase lock is lost

4.3.4 DLL Filter

Since DLL should always be aided by a carrier loop (meaning that there is almost no dynamic stress left to track), it is typically first order, and its noise bandwidth should be very narrow (typically less than 1 Hz in steady state). If one were to use DLL without carrier aiding, its noise bandwidth would have to be increased as there is still dynamic stress left, and the loop would not reach a steady state with narrow noise bandwidth.

4.4 Dynamic Stress Error

This section compares responses of filters (first, second and third-order) to various input signals. This section aims to show why we need to carefully choose the order of individual filters (PLL, DLL, FLL) and what needs to be considered when doing so. There is always some non-zero velocity between the transmitter (satellite) and receiver (user) in the real environment. This has some unpleasant consequences when trying to keep the local replica aligned with the incoming signal.

4.4.1 Error due to Velocity

Consider that there is a constant non-zero velocity between receiver and transmitter. This motion causes a linear rate of change in phase. To correct this, we choose to implement a first-order filter (fig. 4.6). The estimated phase, actual phase and steady state error can be seen in the figure below. The resulting signal is offset by a certain amount. This is caused by dynamic stress and can be evaluated using the formula from table 4.5. The second-order filter (and third-order) does not suffer from this, as can be seen from its response.

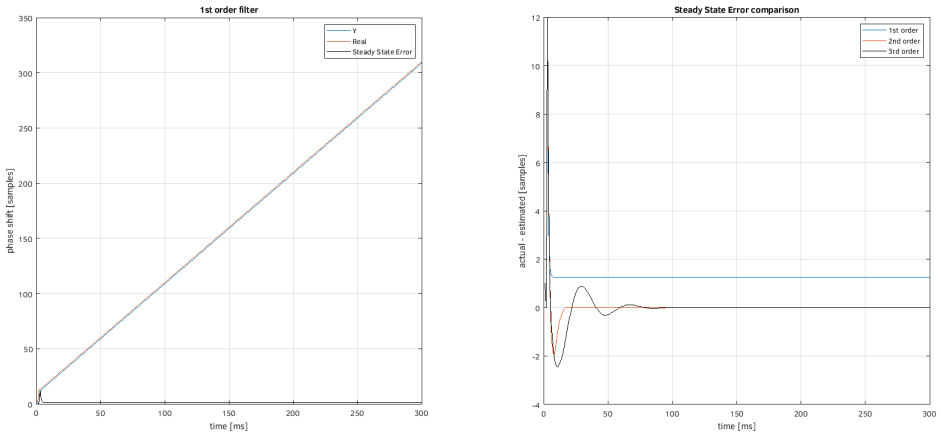


Figure 4.8: 1st order filter response to linear signal

4.4.2 Error due to Acceleration

When the transmitter starts accelerating, the error of the first-order filter starts to grow linearly. The second-order filter is still on track, but this time there is some offset as there was in the previous case for the first-order filter. Similarly, this error can be calculated from table 4.5. Again, the third-order filter has no such offset.

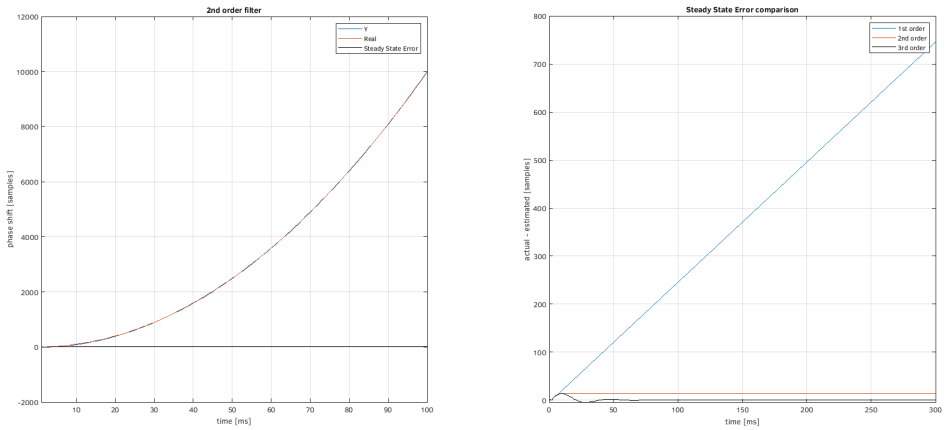


Figure 4.9: 2nd order filter response to quadratic signal

4.4.3 Error due to Jerk

Jerk is telling us how much acceleration is changing in time. This is typically encountered when the transmitter, as well as receiver, are moving. This time, even the second-order filter cannot help us as its error grows linearly (notice that the error of the first-order filter now grows quadratically). Only a third-order filter (which will be off by a certain amount) can handle this dynamic stress.

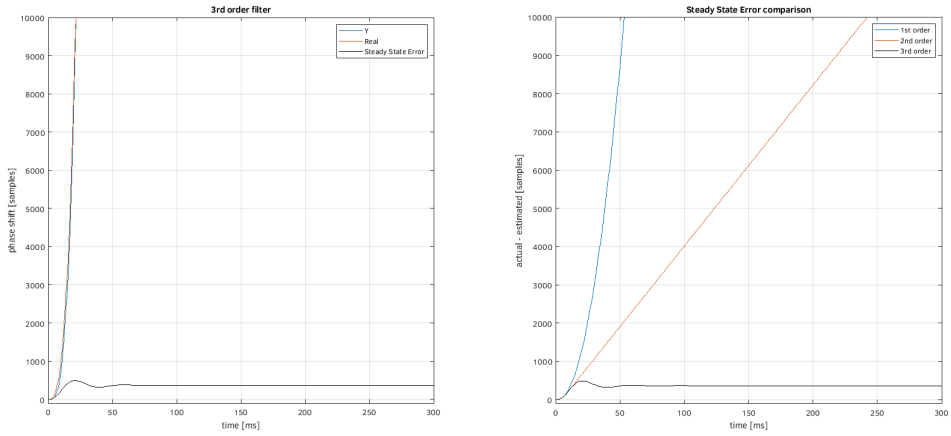


Figure 4.10: 3rd order filter response to cubic signal

Chapter 5

Generating Authentic GNSS Signal

This chapter describes the process behind authentic GNSS signal generation and the motivation behind it. The procedure is not GNSS signal exclusive, but slight modifications may be needed to simulate other signals.

5.1 Motivation

It might not be clear why we would want to generate our own GNSS signal when there is access to real GNSS signals at any point of the day. There are few open-source software-defined radios that demonstrate acquisition available on the internet. Some even have signal tracking and data demodulation, but none generate their own GNSS signal (at least not in their published version), so why bother? For acquisition, this would probably not be worth the trouble. We would be good to go with some simple snippet that generates phase-shifted C/A code on frequency/phase-shifted carrier buried in noise. Only to verify that the acquisition script gets the frequency and prompt right, then straight to working with the real signal.
Nevertheless, developing feedback systems is a much more complicated task. It can be very challenging to find problems in the design even when we have complete control over the signal fed into the system, let alone when we do not, and to top it off, the signal is buried in noise completely.
To design the feedback loops and debug them with ease, we will be generating custom-made signals that we have complete control over.

5.2 Generating Ideal GNSS Signal

In the beginning, we start with the ideal case. No noise, no dynamic stress, only C/A code (and optionally navigation data) modulated onto a carrier wave. We could calculate the pseudo-random sequence every time, but it is much easier to have it stored in a look-up table and load it depending on the specified satellite.
The signal is already generated sampled with provided sampling frequency as it would be found on the output of receiver ADC.
For this ideal scenario, we can generate the signal block by block with the

length of PRN sequence period each, modulate each block onto the carrier wave and store it to a pre-allocated signal vector. This is not such a good idea when dynamic stress is present, which will be discussed later in this chapter. When choosing this approach, we have to make sure that the phase remains continuous throughout the signal. It will have devastating consequences when attempting signal tracking if phase discontinuities are present.

5.3 Observation Model

Before the signal reaches the receiver antenna, it has to propagate through the real environment, picking up noise and other unpleasant properties that make it difficult to process. We create a simple observation model that passes the signal through a channel with *additive white gaussian noise* to approximate this. Multi-path propagation and amplitude scaling is neglected.
The observation model then looks like this

$$x[k] = e^{2\pi j(f_d[k]t[k]+\varphi)} s[\langle k-n \rangle_{\mathrm{mod}\,1023}] + w[k] \tag{5.1}$$

Here φ is some arbitrary phase shift, $\mathbf{f}_d$ is Doppler frequency vector and $\mathbf{w}$ is the AWGN vector.
It is not guaranteed that the sampled signal starts with the first bit of the C/A code, so we shift the signal by some samples before outputting it.

5.3.1 Additive White Gaussian Noise

The amount of noise is specified by C/N_0 [dB-Hz]. First, calculate the power of the signal.

$$P = \frac{1}{N}\sum_{k=1}^{N} \mathrm{s}[k] \cdot \bar{\mathrm{s}}[k] \tag{5.2}$$

Then transform the provided C/N_0 to signal-to-noise ratio.

$$SNR = 10^{\frac{\left(cn0 - 10\log_{10}(\mathrm{f_{sampling}})\right)}{10}} \tag{5.3}$$

Now we have enough information to calculate variance of the noise signal.

$$\begin{aligned} N_0 &= \frac{P}{SNR} \\ \sigma^2 &= \frac{N_0}{2} \end{aligned} \tag{5.4}$$

The variance is half of the noise power because we are dealing with a complex signal. Next, vector of *normally* distributed random numbers is generated and the resulting noise vector takes form of

$$\mathbf{w} = \sigma \cdot (\mathbf{randn} + j \cdot \mathbf{randn}) \tag{5.5}$$

To verify, C/N_0 is calculated from the signal and newly generated noise, and compared to the desired value.

5.4 Adding Dynamic Stress

Due to the physical reality, the transmitted signal suffers from dynamic stress (e.g., there is non-zero mutual velocity between transmitter and receiver as the satellites are constantly moving, causing Doppler shift). Dynamic stress affects not only the carrier wave but C/A code also. Apparent chip frequency changes with variable Doppler frequency, so it must be re-calculated constantly.

$$f_{\text{chip}} = \text{f}_{\text{chipn}} + f_{\text{d}} \frac{\text{f}_{\text{chipn}}}{\text{f}_{\text{carrier}}} \tag{5.6}$$

Where f_{chip} is the apparent chip frequency, f_{chipn} is the nominal chip frequency (1.023 MHz for L1 and E1C) and $\text{f}_{\text{carrier}}$ is frequency of the carrier wave (1.57542 GHz for L1 and E1C).
Note that the Doppler frequency is not constant, and as feedback loops process the signal at millisecond basis, generating the signal 'block-by-block' is not possible (changes of f_{d} would be too sudden). In reality, the Doppler frequency is some continuous function of time. Our simulator approximates it sample by sample, which is sufficient. This is the reason we generate the signal sample by sample. In return, we no longer care about the continuity of the carrier phase as it is always continuous with this approach.

5.5 Implemented GNSS Signals

Developed simulator supports GNSS signals listed in the following table.

	Carrier frequency	Chip frequency	Code length	Code period
GPS L1 C/A	1575.42 MHz	1.023 MHz	1023	1 ms
GPS L5I	1176.45 MHz	10.23 MHz	10230	1 ms
Galileo E1C	1575.42 MHz	1.023 MHz	4092	4 ms

Table 5.1: GNSS signals supported by the simulator

All three are fairly similar. Galileo E1C and GPS L1 C/A have the same carrier and chip frequency, but E1C has four times longer PRN, thus having four times longer code period. GPS L5I has the longest code, it also has the highest chip rate, so the code period is the same as that of L1 C/A.
GPS L1 C/A is the only one not having secondary code. However, secondary codes are of no concern to this thesis so we will not talk about them in great detail. They behave like navigation bits, but we have a priori knowledge of the secondary code sequence.
The biggest difference between these three signals is the modulation. GPS L1

and L5 both use BPSK modulation, and Galileo E1C uses CBOC modulation (we have described BOC and BPSK in chapter 2). Their difference is best seen from their auto-correlation functions. Explaining CBOC is beyond the scope of this thesis, and it can be approached just like BOC with minimum losses.

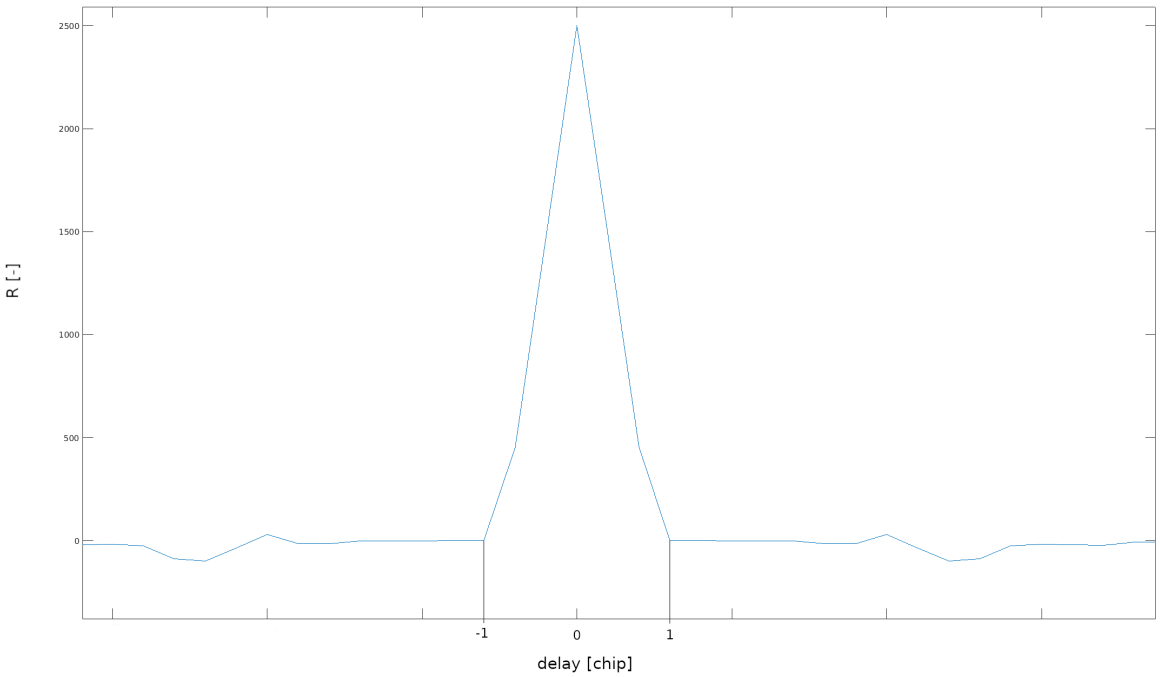


Figure 5.1: BPSK: GPS L1 C/A auto-correlation

The auto-correlation function of BPSK signal is very similar to that of rectangular pulse. This is caused by the correlation properties of C/A code - high correlation when aligned perfectly and almost no correlation when misaligned.
In case of BOC, the auto-correlation function looks a little different. It has more peaks and the main peak is quite narrow compared to BPSK.
We can track BOC signals the same way like BPSK signals by shifting one of the lobes to center frequency. If we do not do this, we have to at least halve the correlator width - the peak of BOC signal's correlation function is narrow compared to BPSK (see figures 5.1 and 5.2).

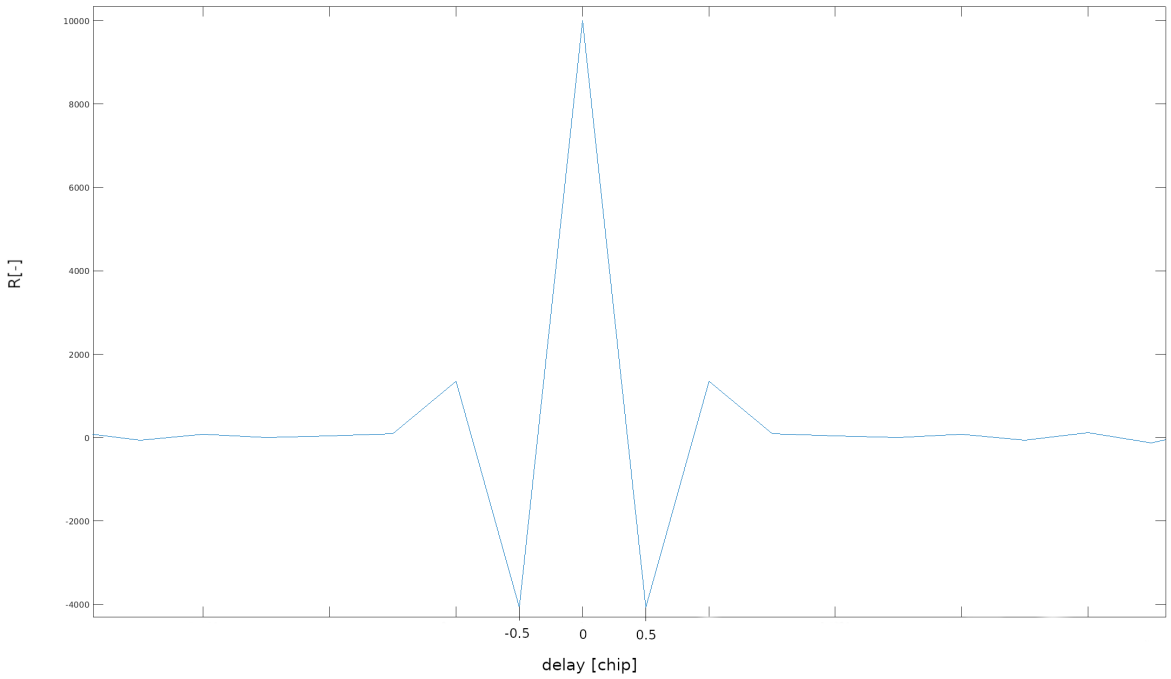


Figure 5.2: BOC: Galileo E1C auto-correlation

Chapter 6

Lock Loops Implementation

This chapter covers the implementation and performance of feedback loops used for carrier and code tracking.

6.1 Lock Loops for Discrete Signals

We need to transform the filters from figure 4.5 to equations that can be used in software implementation. First, replace the continuous integrators with their discrete approximation. We can choose from boxcar accumulator or bilinear transform accumulator, then write the equations using LTI system analysis.

6.1.1 Phase Lock Loop

For second and third-order PLL, the bilinear transform accumulator will be used. The LTI system takes the following form.

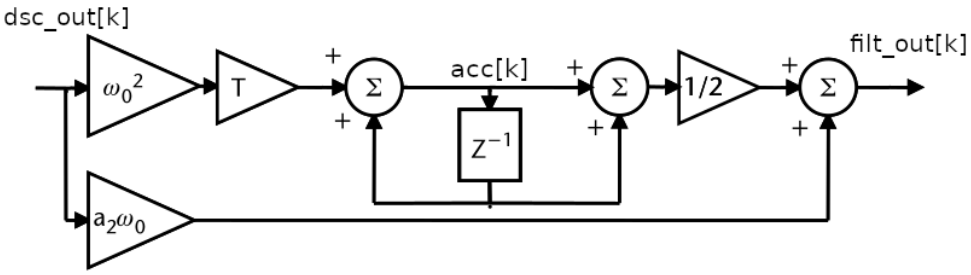


Figure 6.1: Second order PLL using bilinear transform integrator

The *dsc_out[k]* signal is the discriminator output in k-th iteration of the loop. We begin with equation for the only accumulator (*acc[k]*).

$$\mathrm{acc}[k] = \omega_0^2 T \mathrm{dsc\_out}[k] + \mathrm{acc}[k-1] \tag{6.1}$$

Here ω_0 is the natural frequency of the filter, determined from table 4.5 and T is the time interval between samples. This interval varies depending

on coherent accumulation length.
For the filter output (*filt_out[k]*) we get

$$\text{filt\_out}[k] = \frac{1}{2}\left(\text{acc}[k] + \text{acc}[k-1]\right) + a_2\omega_0\text{dsc\_out}[k] \tag{6.2}$$

The filter output is then fed back to the carrier NCO like this

$$\text{f\_est} = \text{f\_basis} + \text{filt\_out}[k] \tag{6.3}$$

Where *f_basis* is frequency detected by acquisition and *f_est* is used in the next iteration to generate carrier replica.

$$\text{ex} = \cos\left(2\pi \cdot \text{f\_est} \cdot \text{t} + \text{phase\_est}\right) - \text{j}\sin\left(2\pi \cdot \text{f\_est} \cdot \text{t} + \text{phase\_est}\right) \tag{6.4}$$

To ensure continuous phase throughout iterations, *phase_est* is calculated from the argument of the goniometric functions.
Similarly for third order PLL we get

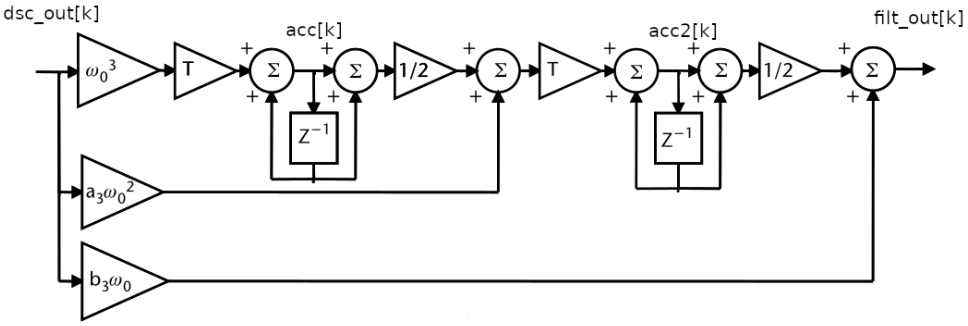


Figure 6.2: Third order PLL using bilinear transform integrator

One additional equation is needed for the third order filter.

$$\begin{aligned}
\text{acc}[k] &= \omega_0^3 T\,\text{dsc_out}[k] + \text{acc}[k-1] \\
\text{acc2}[k] &= T\left(\frac{1}{2}\left(\text{acc}[k] + \text{acc}[k-1]\right) + a_3\omega_0^2\text{dsc_out}[k]\right) + \text{acc2}[k-1] \\
\text{filt_out}[k] &= \frac{1}{2}\left(\text{acc2}[k] + \text{acc2}[k-1]\right) + b_3\omega_0\text{dsc_out}[k]
\end{aligned} \tag{6.5}$$

The first accumulator is sometimes called *acceleration accumulator* and the second one *velocity accumulator*. As the name suggests, estimated velocity (acceleration) is stored in the first (second) accumulator. This is very useful for verifying that the loop is working as expected.

6.1.2 Delay Lock Loop

The delay lock loop is implemented identically to the phase lock loop. The only difference is how the filter output is fed back to the code NCO.

The phase lock loop is used to refine the estimate of the carrier frequency, whereas the delay lock loop refines the estimate of the code frequency. As frequency basis, we will now use the *nominal chip frequency*, to which we add the output of the DLL filter. Consequently, the code period varies across iterations, meaning there needs to be a re-sampling of the code replica done in each iteration of the tracking loop. Again as in the case of PLL, the code NCO implemented in software does not have the luxury of over-flowing hardware registers. Thus, the continuity of the code phase (displacement of the prompt) must be tracked manually.

6.1.3 Frequency Lock Loop

Since the FLL discriminator outputs frequency error (unlike PLL/DLL, which produces phase error), the design requires one extra integrator. In return, the filter will usually be one order lower than that of PLL. Even in cases of extreme dynamic stress, we do not expect the rate of change of frequency to be higher than quadratic, meaning the second-order FLL filter should be able to withstand it.

Chapter 7

Lock Loop Testing

We have discussed general feedback systems and their performance under varying dynamic stress. This chapter aims to test the implemented lock loops on actual GNSS signals - from our custom generator, generated with laborathory signal generator and live GNSS signals recorded on SDR while working on this thesis.

7.1 Loop Testing with Custom GNSS Signal

This section explains basic setups to test feedback loops most efficiently. Before testing the whole system, we want to test individual loops separately, which is not possible with real signal.
Carrier tracking requires the code to be wiped off. However, this is not possible without DLL when the Doppler shift is present. We can easily prevent that by locking chip frequency on its nominal value, resulting in replica code being perfectly aligned at all times. Now we are free to test PLL/FLL with an arbitrary course of Doppler frequency function and phase bias.
On the other hand, code tracking requires the carrier to be removed. This is achieved simply by replacing the carrier with 1 in signal generation, equivalent to having an ideal frequency and phase synchronization. At this point, nothing stops us from testing DLL with dynamic stress affecting only the C/A code. The pre-detection integration time (PDI) in the following tests is always set to one code period (as it would be before bit synchronization). We can do this since tracking with longer PDI is essentially the same. The only difference is in the discriminator used (we use discriminators insensitive to bit transition before the synchronization).

7.2 First Order Filters

To test first-order DLL, let us start with a simple scenario - constant Doppler frequency $f_d = 510$ Hz, code phase bias of 3 samples and assuming carrier is wiped off.
There is a constant steady-state error (fig. 7.1) even for constant Doppler

frequency. This is because DLL is tracking phase, not frequency. We know that $f = \frac{\mathrm{d}\varphi}{\mathrm{d}t}$. For constant frequency, the phase changes linearly. According to section 4.4, a constant steady-state error is expected. The loop is unable to keep the local replica aligned with incoming code, resulting in the late version having higher energy, as can be seen from the correlator (fig. 7.2).

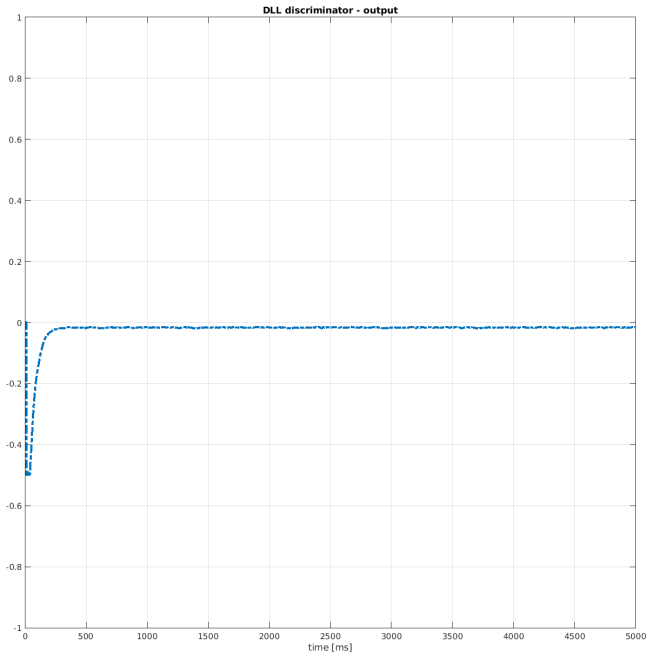


Figure 7.1: DLL Discriminator: $f_d = 510$ Hz, 1st order filter

When the Doppler frequency starts to grow linearly, so does the error (fig. 7.3) as expected from theoretical background. This scenario is beyond the capabilities of first-order DLL, and we have to switch to second order. Before we move to that, we will run similar tests for the carrier tracking loop.
The signal set-up remains the same ($f_d = 510$ Hz), but assuming code wipe-off this time, and it is the first order FLL we are testing.

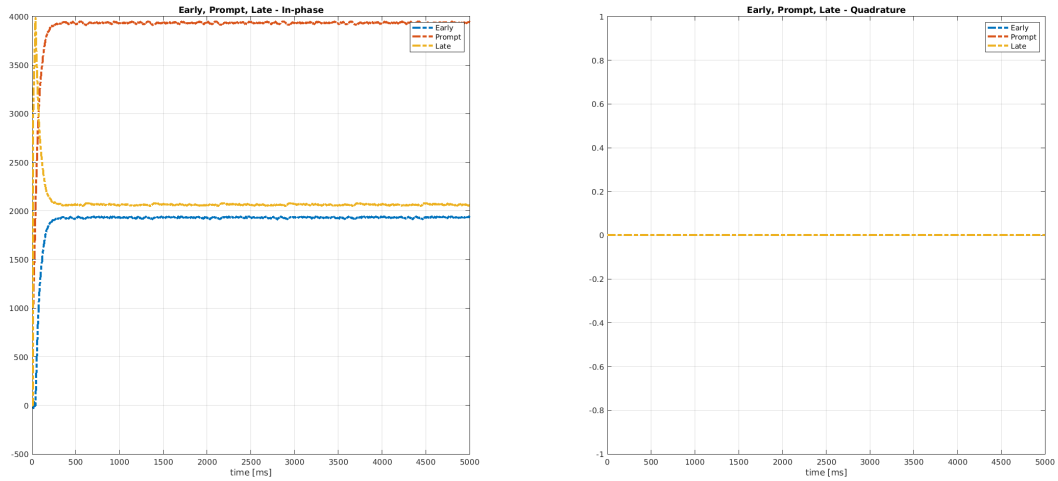


Figure 7.2: Correlator: $f_d = 510$ Hz, 1st order DLL

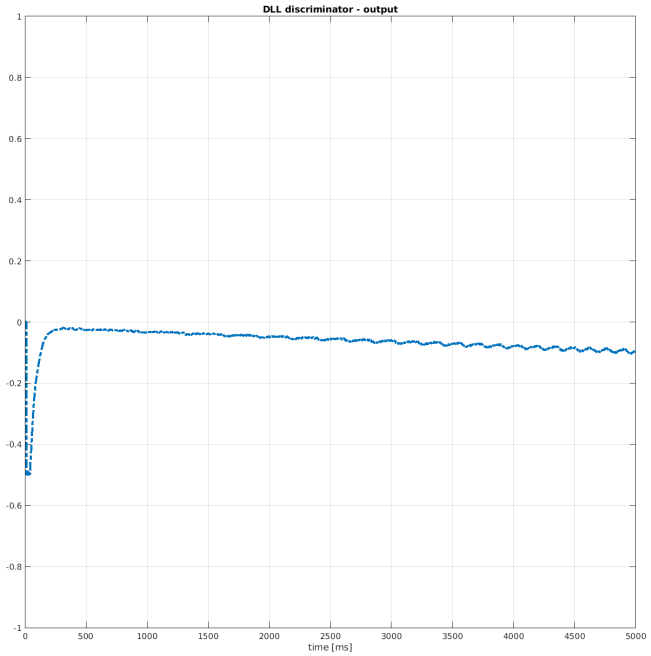


Figure 7.3: DLL Discriminator: f_d growing linearly, 1st order filter

The constant frequency offset is not causing any steady state error (fig. 7.4). Since FLL directly integrates frequency (not phase as in the case of DLL and PLL), its filter response is equivalent to constant input.

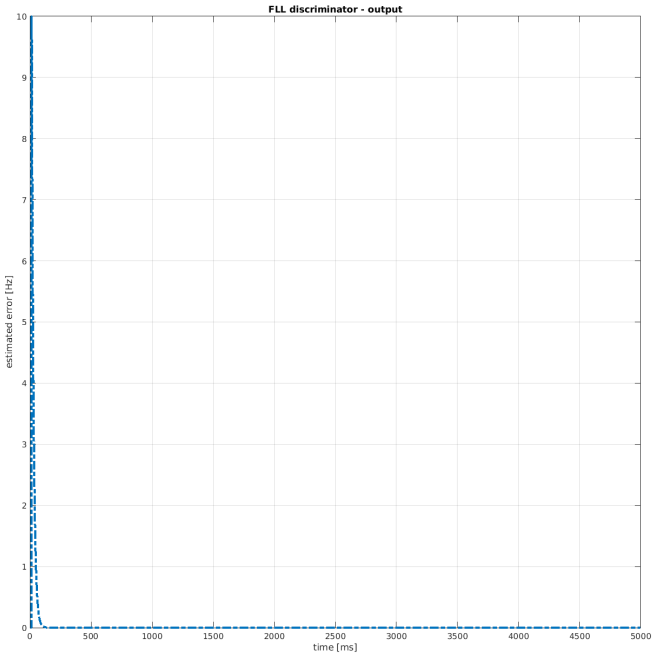


Figure 7.4: FLL Discriminator: $f_d = 510$ Hz, 1st order filter

At first glance, the correlator (fig. 7.5) output does not seem right. This has a simple explanation, FLL synchronizes frequency, not phase. We can imagine the constellation points rotating until the lock is reached due to frequency offset. When FLL locks, the constellation points remain at their position, but it is not guaranteed that said position is on the real axis.

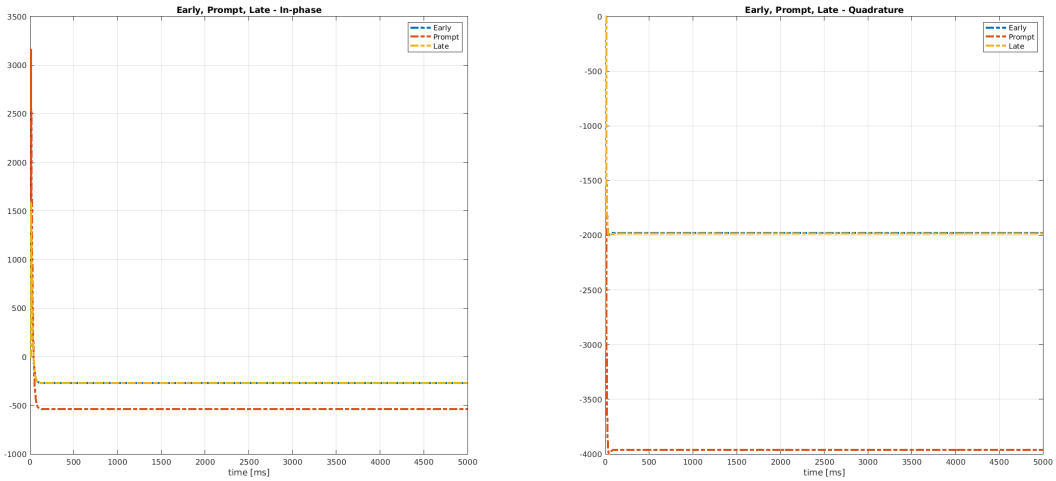


Figure 7.5: Correlator: $f_d = 510$ Hz, 1st order FLL

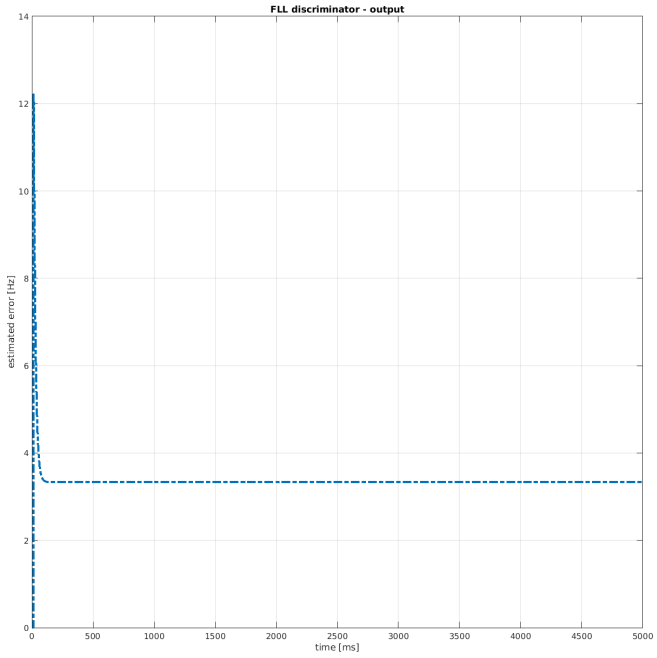


Figure 7.6: FLL Discriminator: f_d growing linearly, 1st order filter

We can see the expected steady state error after adding a linear profile to

the Doppler frequency (fig. 7.6). Consequently, the energy in the correlator flows from the in-phase to the quadrature branch as there is still some constant frequency offset (fig. 7.7).

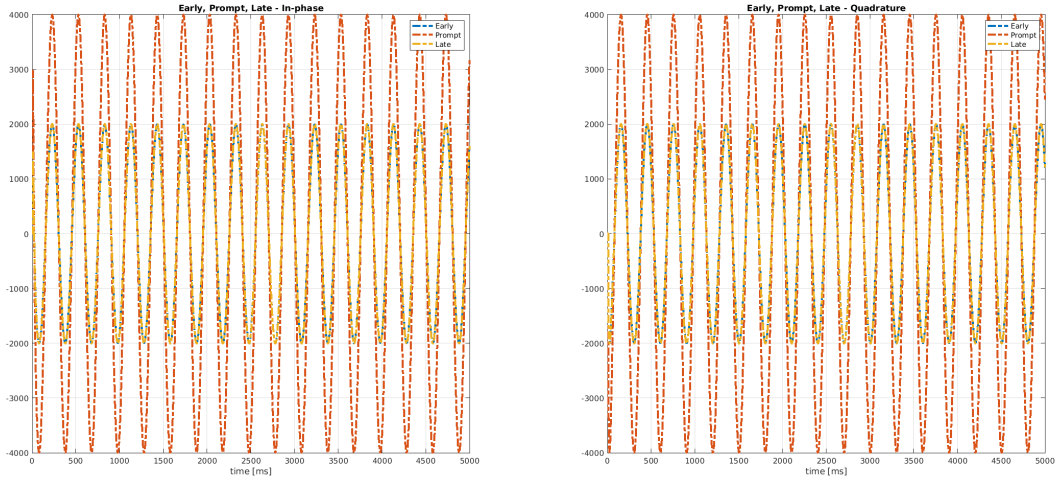


Figure 7.7: Correlator: f_d growing linearly, 1st order FLL

When testing PLL, we expect similar results to DLL as, again, it is the phase we are tracking (only the phase of the carrier wave this time). We have to be much more careful with Doppler frequency, high frequencies are causing rapid phase changes, and we could quickly end up outside of the pull-in region, rendering the loop useless.
With this in mind, we choose $f_d = 5$ Hz at first and a carrier phase bias of $\pi/4$ radians. Again, we can see a constant steady state error (fig. 7.8) due to the linear rate of change of the carrier phase. Now we add the constant rate of change to f_d, simulating constant acceleration. Expectedly, the steady-state error now grows linearly (fig. 7.9).

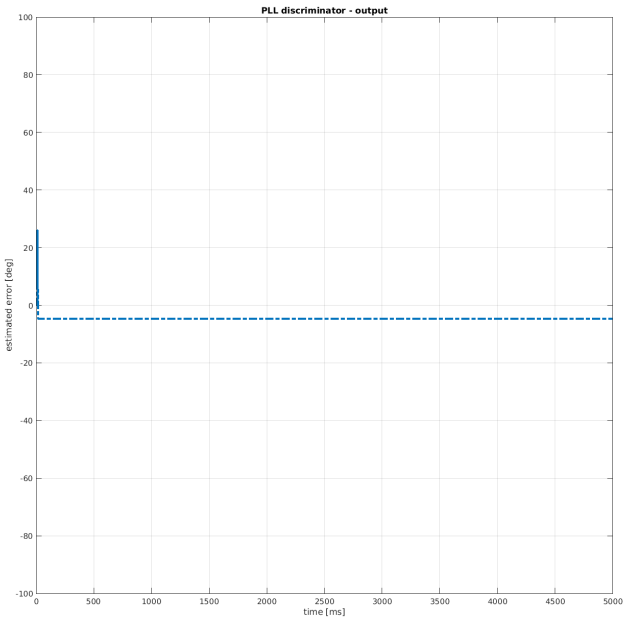


Figure 7.8: PLL Discriminator: $f_d = 5$ Hz, 1st order filter

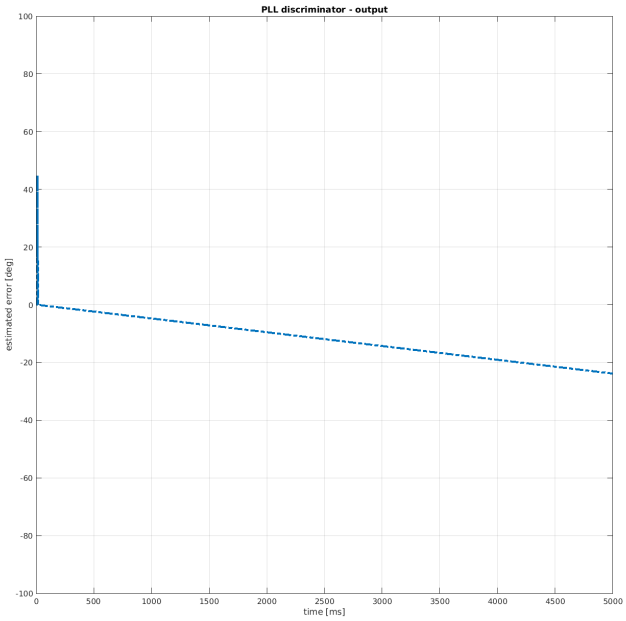


Figure 7.9: PLL Discriminator: f_d growing linearly, 1st order filter

7.3 Second Order Filters

Upgrading to second-order filters should bring better performance under increasing dynamic stress. Starting this time with FLL, our interest focuses on filter response rather than the discriminator itself.
After the steady state is reached, we can see (fig. 7.10) that the frequency offset (due to acquisition resolution) is 20 Hz. After all, the Doppler frequency for this example was set to 520 Hz, and the acquisition script estimated 500 Hz. Another piece of information is given to us by the accumulator. It stabilizes on zero after reaching the steady state, meaning the Doppler frequency is constant (there is no acceleration).

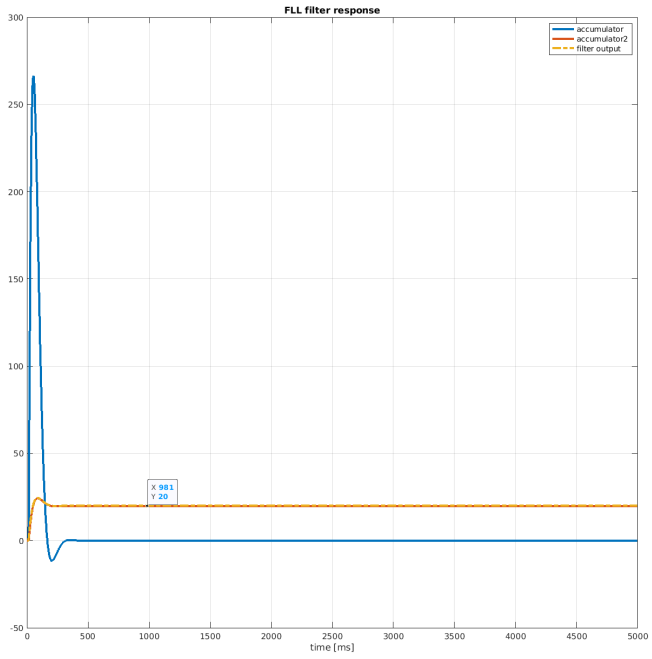


Figure 7.10: FLL: $f_d = 520$ Hz, 2nd order filter

However, carefully look at what happens when we set $f_d = 0$ Hz and let it grow linearly. The filter output (fig. 7.11) tells us how the mutual velocity developed in time. From the accumulator, we see that the rate of change in frequency was 100 Hz/s. Notice that the discriminator shows no steady state error (fig. 7.12) as second-order FLL has no problem even when accelerating occurs.

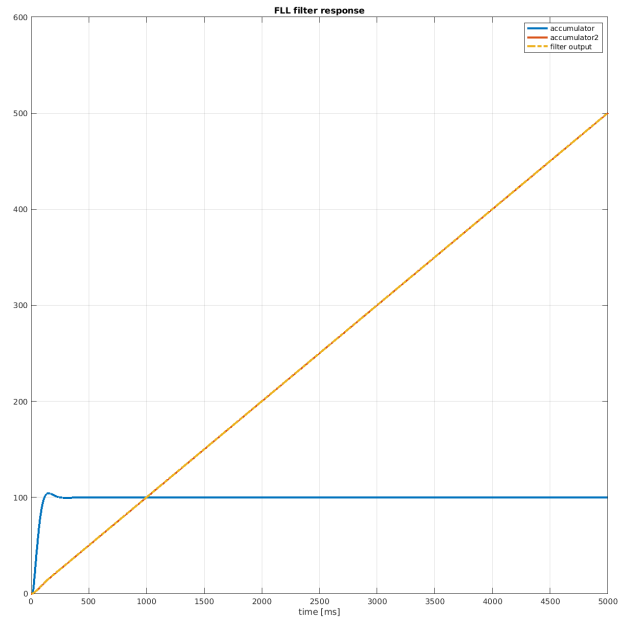


Figure 7.11: 2nd order FLL filter: $f_{\text{rate}} = 100$ Hz/s

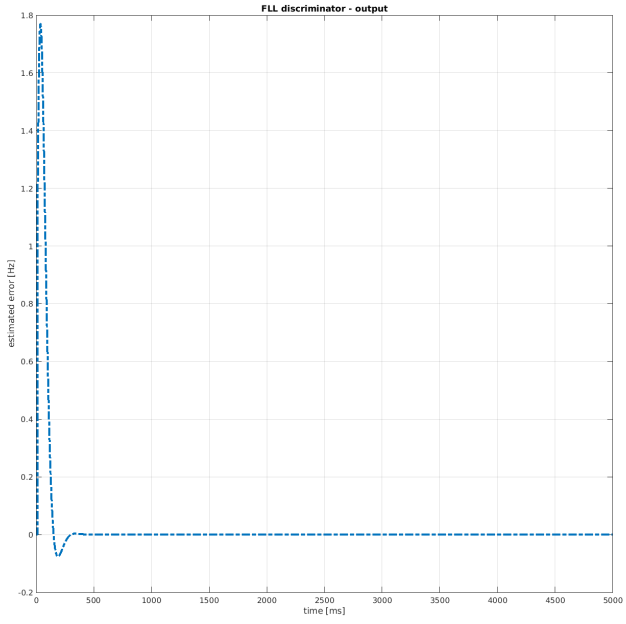


Figure 7.12: FLL Discriminator: $f_{\text{rate}} = 100$ Hz/s, 2nd order filter

Loops that are tracking phase (be it code phase or carrier phase) will perform slightly worse in a sense that there will be constant steady state

error for acceleration scenario because phase has quadratic growth here.
To no surprise, both DLL (7.14) and PLL (fig. 7.13) perform as predicted. If we want to remain synchronized with incoming signals in scenarios comparable to our test environment, we are forced to choose third-order lock loops. They come with their own limits and unpleasant properties, as discussed in the following section.

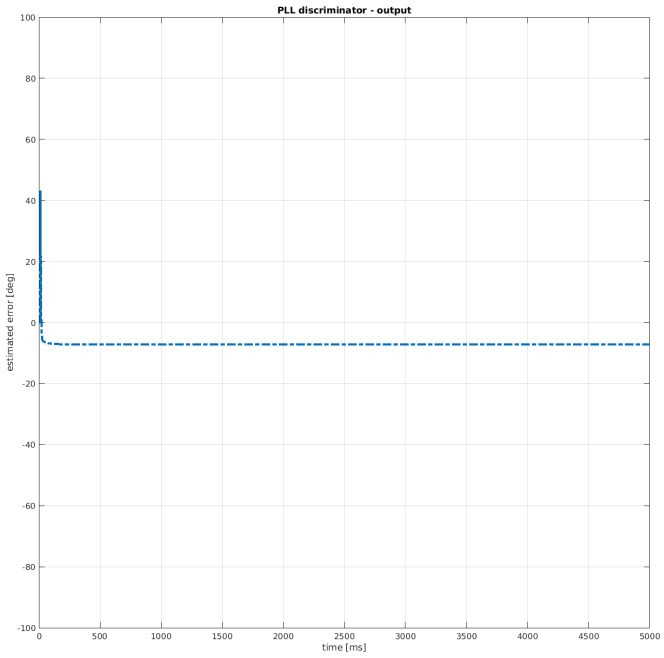


Figure 7.13: PLL Discriminator: f_d growing linearly, 2nd order filter

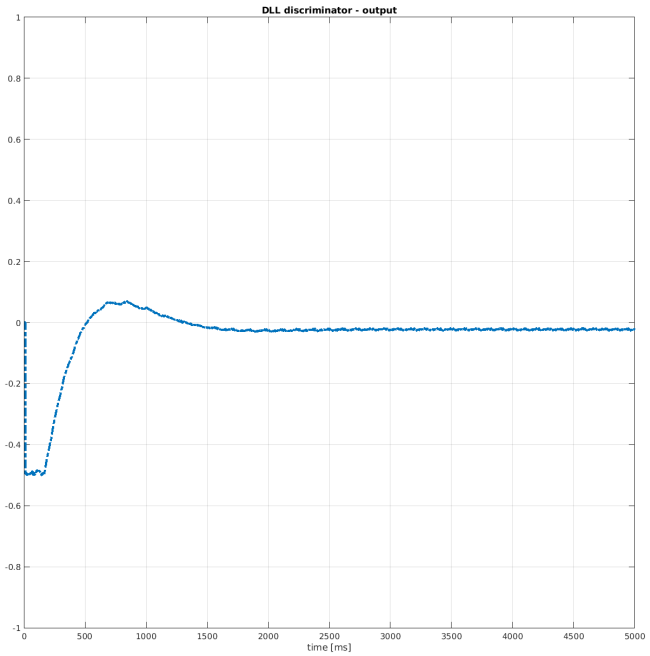


Figure 7.14: DLL Discriminator f_d growing linearly, 2nd order filter

7.4 Third Order Filters

When dealing with strong dynamic stress, only third-order filters usually can withstand it without steady state error. However, this comes at a cost; otherwise, we would only use third-order filters over anything else. As demonstrated later in this section, they take much more time to stabilize, and their transient response is very abrupt. If the noise bandwidth is not chosen carefully, the initial response can massively overshoot, and steady state might not be reached.
Using the same set-up as for the second-order DLL, there is no steady state error (7.16), although it took nearly 15 seconds for the loop to reach steady state (fig. 7.15). One could argue that raising noise bandwidth would yield faster stabilization. On the contrary, this is a double-edged sword when noise is present (this will be the subject of a later section in this chapter).

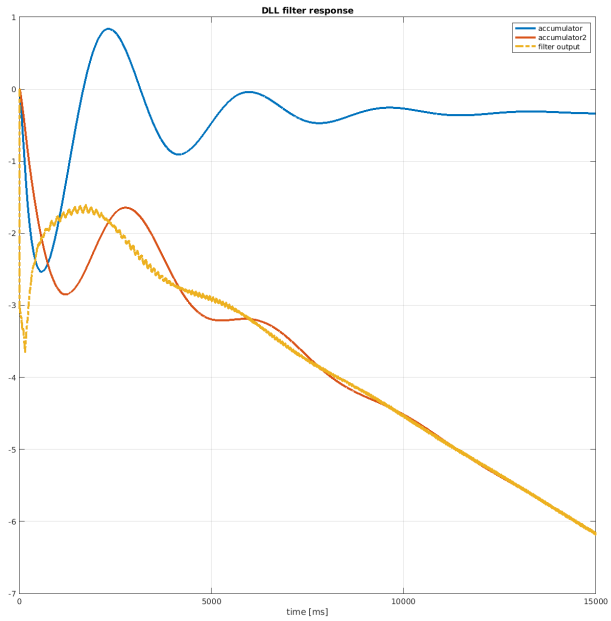


Figure 7.15: 3rd order DLL filter: f_d growing linearly

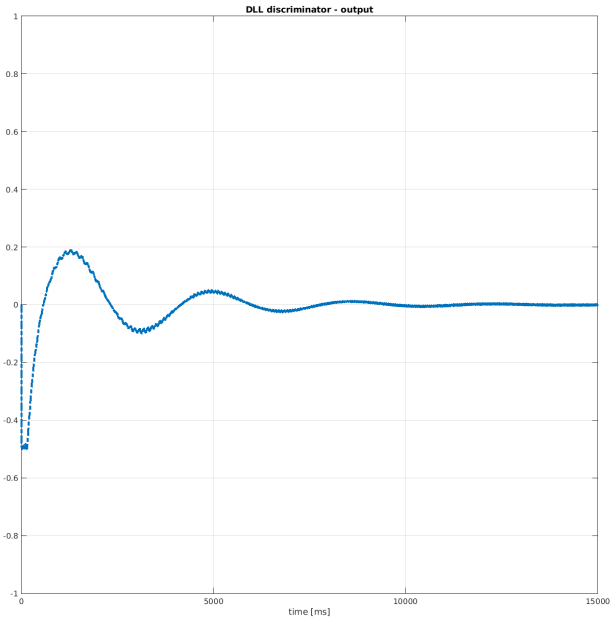


Figure 7.16: DLL Discriminator f_d growing linearly, 3rd order filter

7.5 Assisted Loops

As mentioned in section 4.3.3, sometimes we cannot afford both PLL and FLL in our design. FLL-assisted-PLL brings robustness of FLL and precision of PLL. For demonstration purposes, let us set $f_d = 1510$ Hz and carrier offset $\pi/4$ radians. There is no phase offset after the loop reaches steady state, so maximum energy is concentrated in the in-phase branch (7.17), allowing precise demodulation.

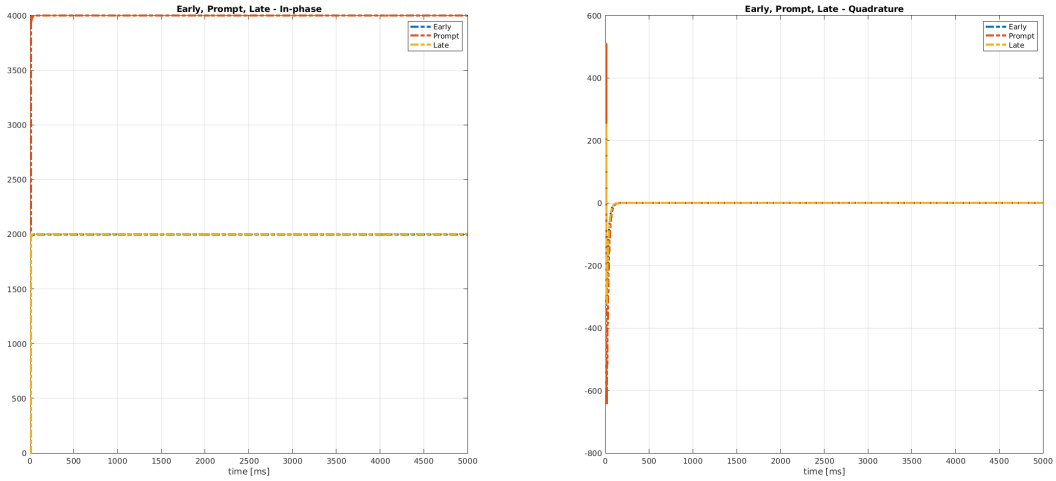


Figure 7.17: Correlator: $f_d = 1510$ Hz, FLL-assisted-PLL filter

7.6 Noise Bandwidth

The filter's noise bandwidth (B_n) affects two crucial things - how fast the steady state is reached and how much noise is let in. We will try to show now why it is a tough compromise to make when choosing the right B_n for our filter. Second-order FLL ($f_d = 1550$ Hz, 50 Hz/s rate) stabilizes almost immediately when we set $B_n = 100$ Hz, but look what happens when we add some noise to our generated signal (fig. 7.18).
Lowering the B_n to just 5 Hz gives us more precise result. However it takes a while for the loop to reach steady state (fig. 7.19).

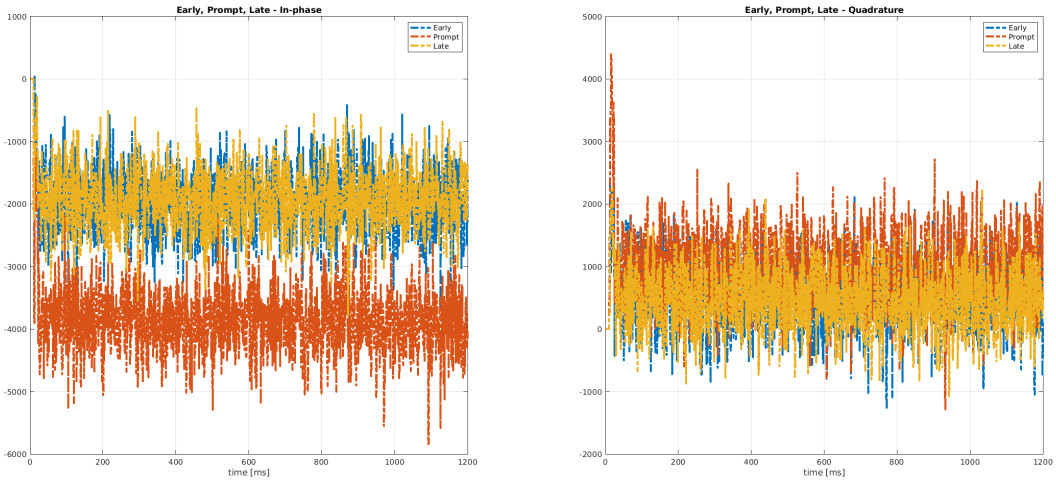


Figure 7.18: Correlator: SNR = -21 dB, B_n = 100 Hz

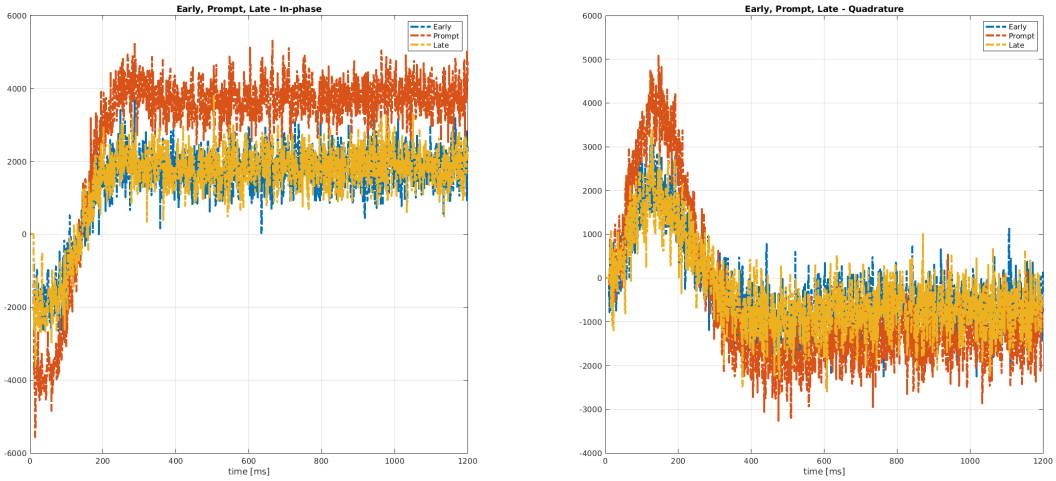


Figure 7.19: Correlator: SNR = -21 dB, B_n = 5 Hz

Another important property of noise bandwidth is that B_n is inversely proportional to steady state error (i.e., decreasing B_n increases steady state

error). Decreasing B_n to 10 Hz should increase steady state error to 1.25 Hz, which is exactly what happens when we do so (figs. 7.20 and 7.21).
Choosing noise bandwidth is no easy task. It heavily depends on the operating environment, filter order, and even what we are tracking (FLL, PLL and DLL have different typical values of B_n. Depending on the B_n value, we refer to the filter as wide, narrow, or very narrow.
Steady state error can be calculated from 4.5. Relation between B_n and steady state error is hidden inside filters' natural frequency ω_0, which might not be evident immediately.

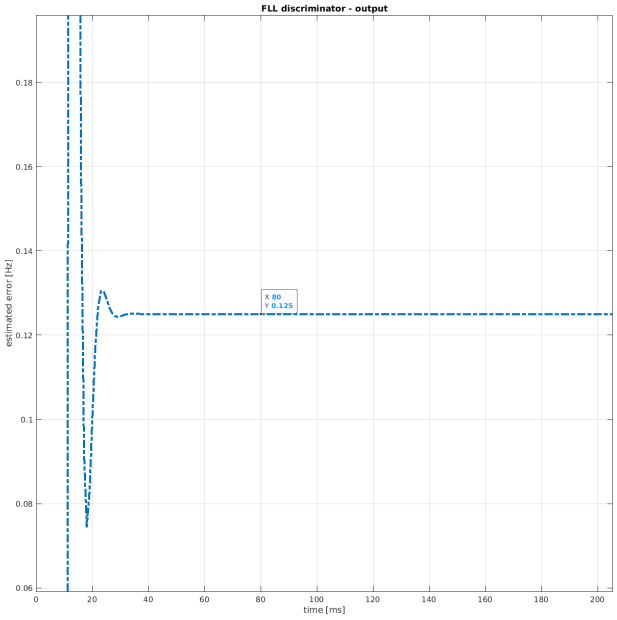


Figure 7.20: $B_n = 100$ Hz: 1st order steady state error

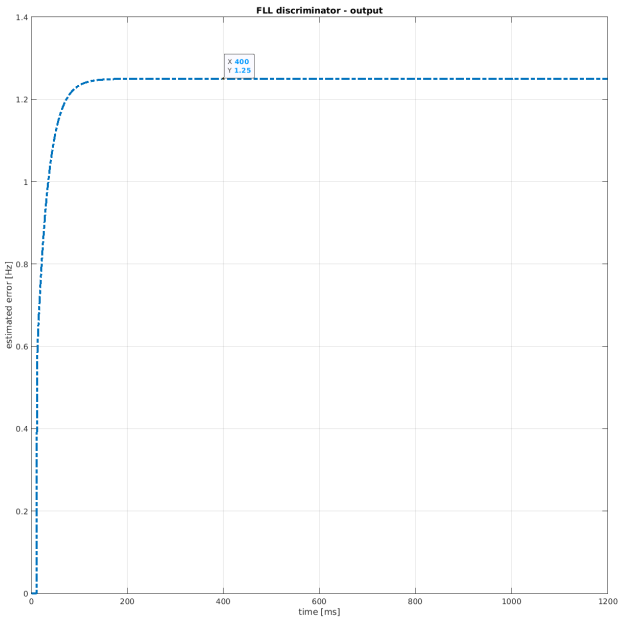


Figure 7.21: $B_n = 10$ Hz: 1st order steady state error

7.7 Ordinary vs. Costas PLL

This section serves as a quick explanation of why the Costas loop is needed before bit synchronization. Figures 7.22 and 7.23 show how ordinary vs. Costas PLL perform with navigation data present.
Ordinary and Costas discriminators are listed in tables 4.1 and 4.2. As stated in section 4.1, we cannot use ordinary PLL before bit synchronization. The navigation data bit or secondary code bit causes a 180°phase shift. When using ordinary PLL, the shift makes it concentrate energy into the quadrature branch instead of the in-phase branch. For Costas PLL, only the sign of changes, but the energy is still concentrated in the in-phase branch.
After bit synchronization, we can switch to ordinary PLL, which has the benefit of not suffering squaring loss [2, p. 476]. The coherent integration time is then increased, but the process remains the same in principle. This is why we did not implement bit synchronization in this thesis.

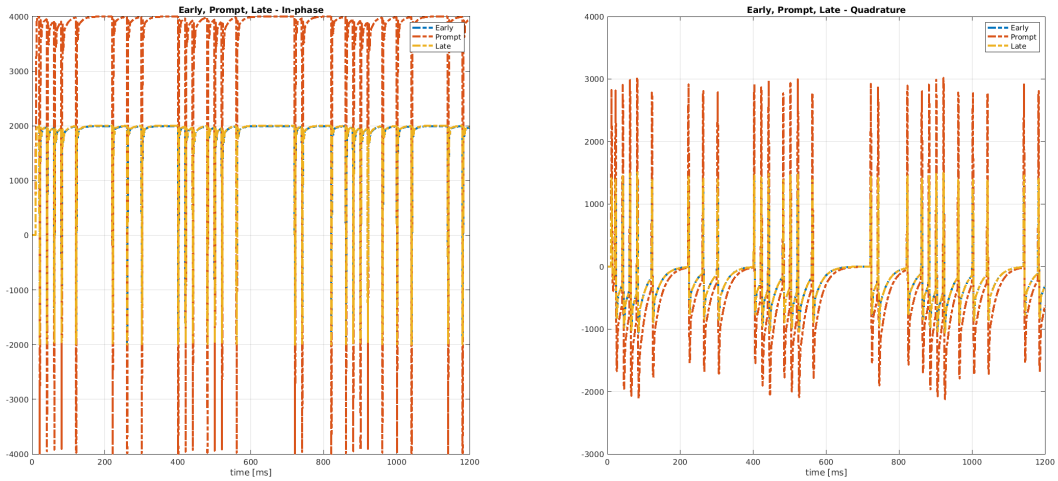


Figure 7.22: Correlator: Ordinary PLL

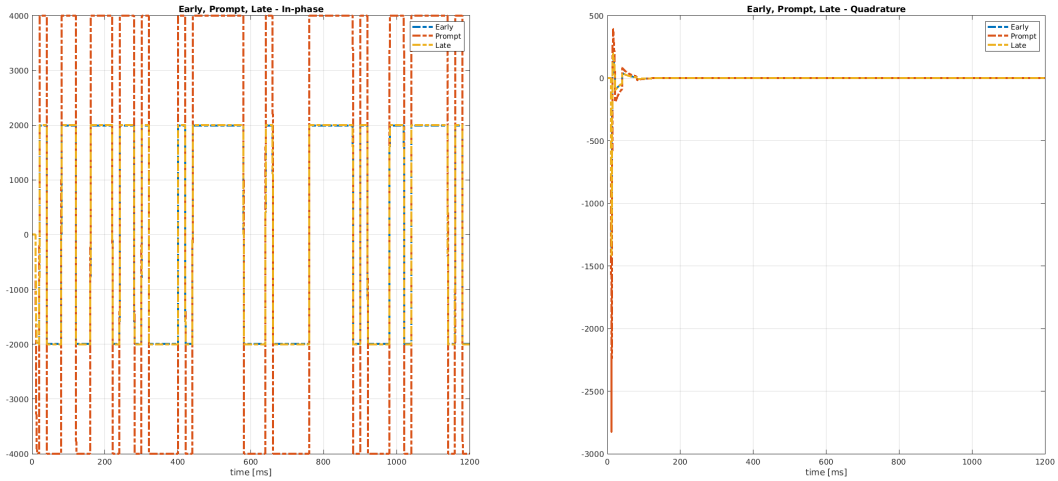


Figure 7.23: Correlator: Costas PLL

7.8 Tracking Live GNSS Signal

We have designed the lock loops, and have tested them on signals generated from our simulator and from laborathory generator (which is not documented in this thesis as it is not that interesting compared to tracking real GNSS signal).
The signal was recorded on May 10th at 14:33:28 local time (UTC+2). Satellites shall be chosen according to Trimble sky plot [*https://www.gnssplanning.com/#/skyplot*].

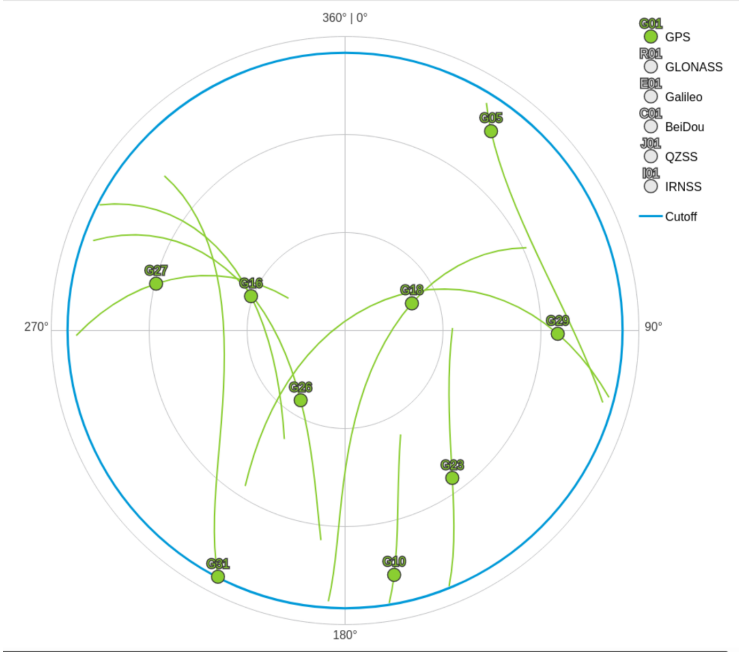


Figure 7.24: Trimble Sky Plot: GPS

First satellite chosen is G18, because it is the closest one to zenith and should be easy to track (less dynamic stress compared to satellites closer to horizon). The lock loop filter set-up can be found in table 7.1.

	Order	B_n [Hz]
DLL	2	2
PLL	x	x
FLL	2	5

Table 7.1: Lock loop filters settings: GPS 18

As can be seen from figures 7.25 and 7.26 the loop stabilizes after around one second.

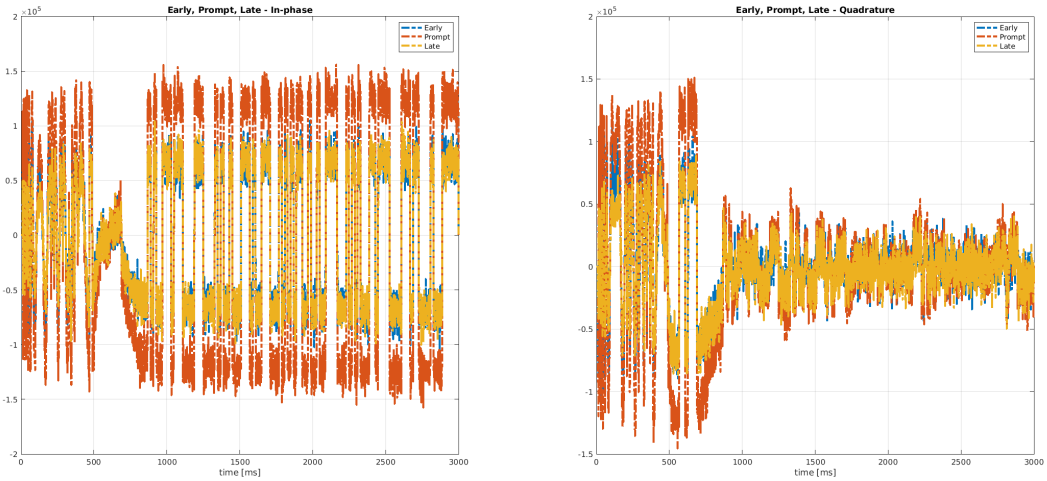


Figure 7.25: Correlator: Live Signal - Zenith

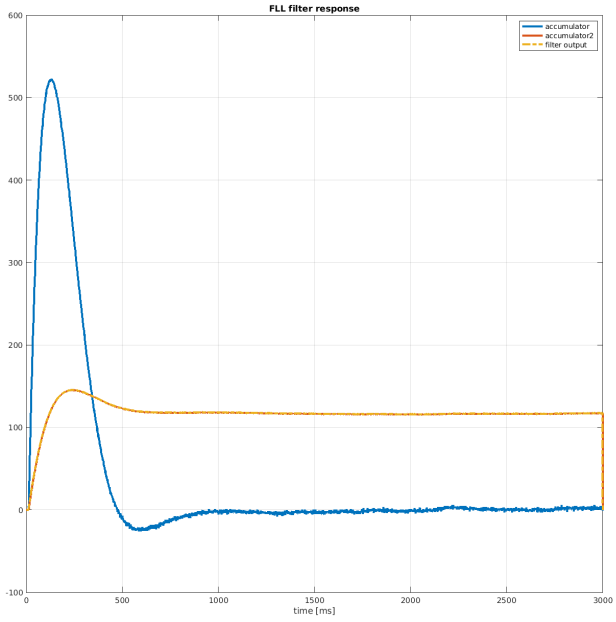


Figure 7.26: 2nd order FLL filter: Live Signal - Zenith

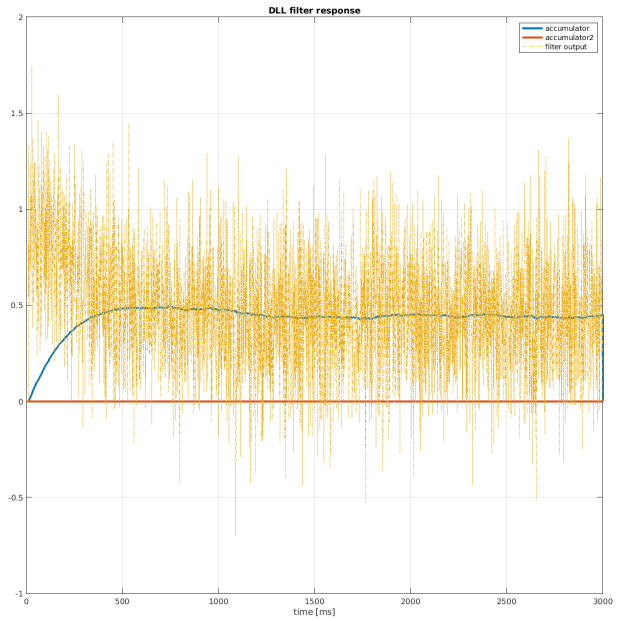


Figure 7.27: 2nd order DLL filter: Live Signal - Zenith

Next we choose satellite that is closer to horizon. For satellite number 10, the Doppler frequency should be much higher.

	Order	B_n [Hz]
DLL	2	2
PLL	2	15
FLL	1	15

Table 7.2: Lock loop filters settings: GPS 10

This time we have used FLL-assisted-PLL, and as can be seen from 7.28, it has no problem to track a satellite even if it is close to horizon.

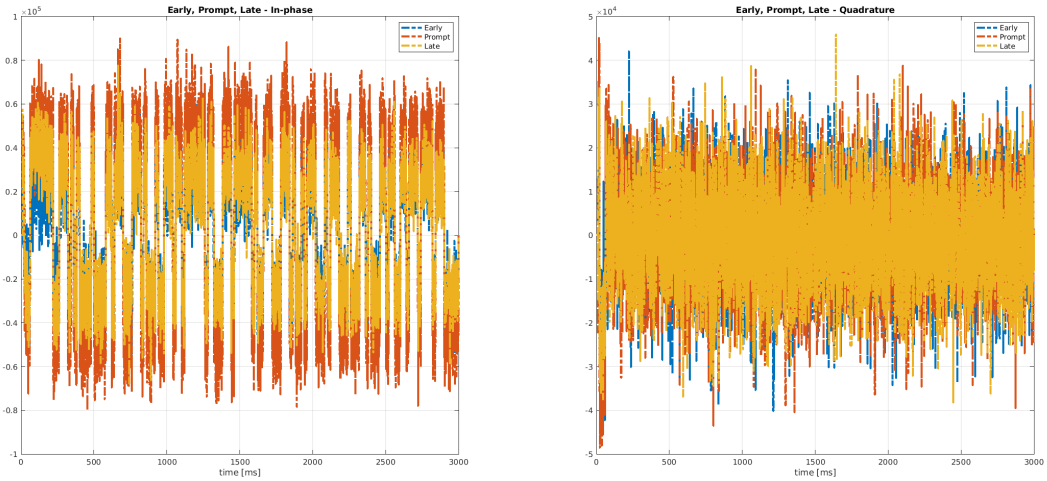


Figure 7.28: Correlator: Live Signal - Horizon

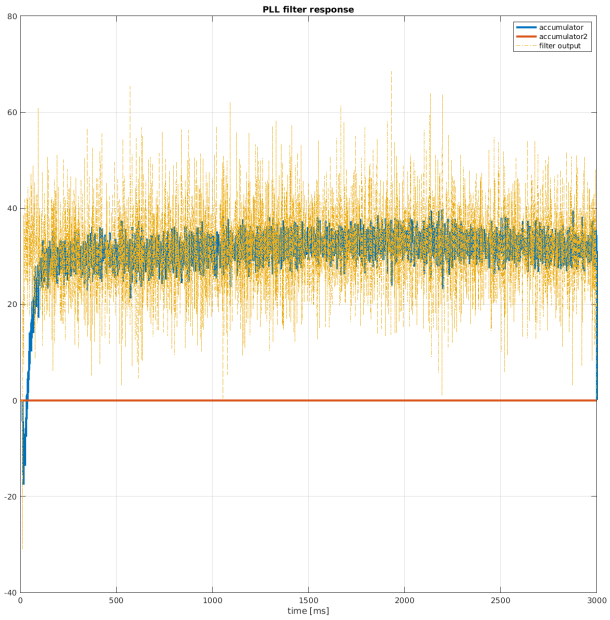


Figure 7.29: FLL-assisted-PLL filter: Live Signal - Horizon

We can use the same recording to track Galileo E1C. Again, we will use Trimble Sky Plot to choose a visible satellite.

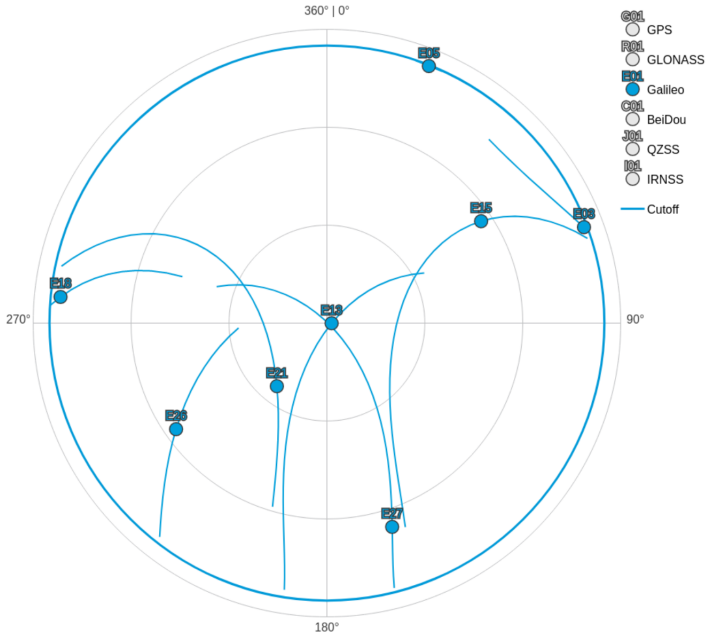


Figure 7.30: Trimble Sky Plot: Galileo

The satellite number 13 is almost directly above our antenna. We could use this to test PLL without any assistence. The Doppler frequency must be low, so we can searcher shorter frequency range with denser grid. Resulting acquisition estimate will be more precise, giving us better chance to land inside PLL's pull-in range.

	Order	B_n [Hz]
DLL	2	2
PLL	3	15
FLL	x	x

Table 7.3: Lock loop filters settings: Galileo 13

The loops lock effortlessly after few hundred milliseconds. We can see the quality of the received signal from figure 7.31. It is due to the fact that the signal has to travel the shortest distance through atmosphere compared to the GPS recordings. Also, there are basically no obstacles as the satellite is directly above the antenna.

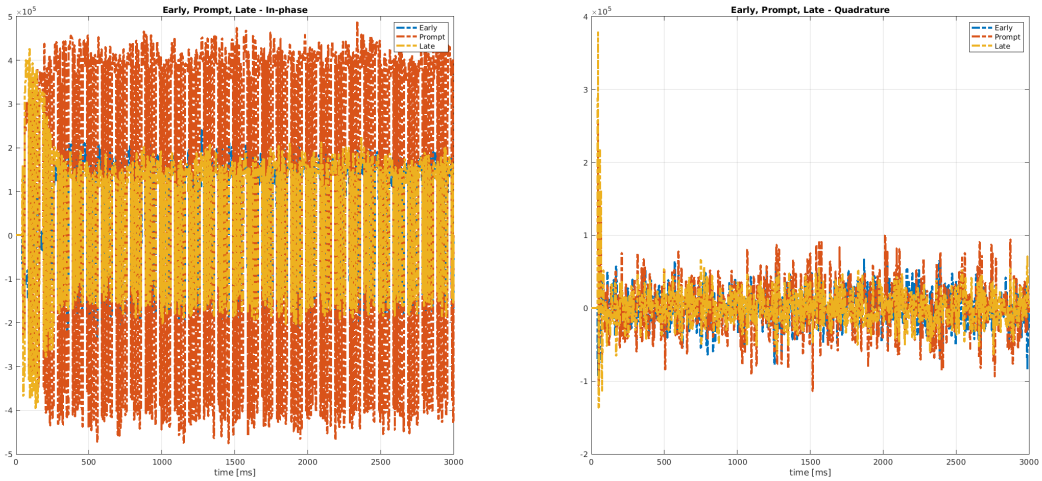


Figure 7.31: Correlator: Live Signal - Galileo 13

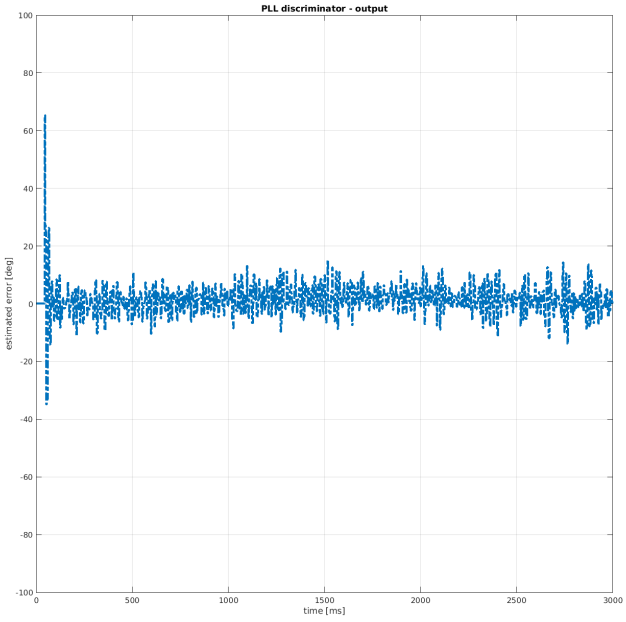


Figure 7.32: PLL discriminator: Live Signal - Galileo

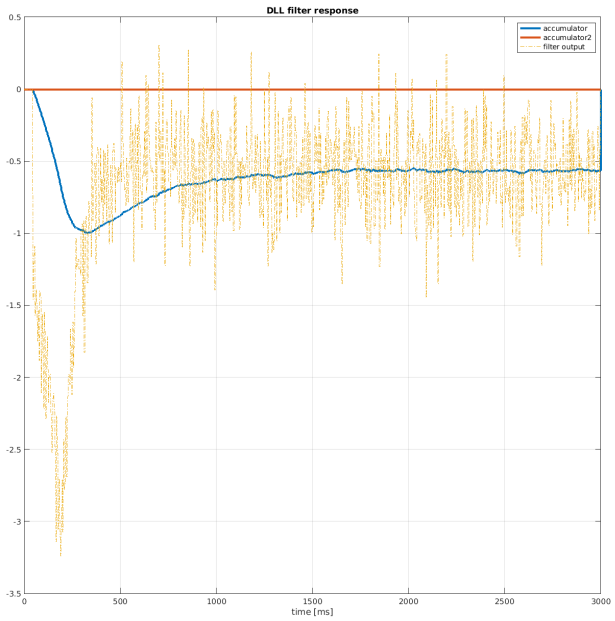


Figure 7.33: 2nd order DLL filter: Live Signal - Galileo

Chapter 8

Conclusion

In this thesis, we have described feedback systems used in GNSS receivers. We have illustrated how general feedback systems work and have used it as a benchmark in a later chapter, where implemented lock loops have been tested. Chapter 5 described the process behind a generation of authentic GNSS signals that have been used while implementing the lock loops. In chapter 7, we have tested the implemented lock loops and verified that their behavior is in accordance with the general feedback system. We have utilized the custom GNSS signal to correctly design these loops and moved to tests on signals generated from laboratory generator as well as real GNSS signals for comparison.
Developed Matlab simulator supports authentic signal generation of GPS L1 C/A, GPS L5I and Galileo E1C signals. User can set-up almost any combination of tracking loops, even assistance between them. There are numerous discriminators implemented and user can switch between them even mid program. All relevant information is either printed to console or plotted and saved.

Appendix A

SDR Source Code

```
gnss
├── figures
│   ├── thesis
│   │   └── exported_figures.zip .2 src
│   ├── acq.m
│   ├── codes_E1C.mat
│   ├── codes_L1CA.mat
│   ├── codes_L5I.mat
│   ├── discriminators.m
│   ├── filters.m
│   ├── initTracking.m
│   ├── rtz.m
│   ├── sdr.m
│   ├── signalGen.m
│   └── tracking.m
└── README.md
```

Up-to-date version of the source code can be found here: `https://gitlab.fel.cvut.cz/nikolana/gnss`

Appendix B

Bibliography

[1] F. Vejražka, P. Pánek, Z. Hrdina. *Rádiové určování polohy.* [*Družicový systém GPS*] České Vysoké Učení Technické v Praze, Fakukta Elektrotechnická, 1995.

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

[3] K. Borre, D.M. Akos, N. Bertelsen, P. Rinder, S.H. Jensen. *A Software-Defined GPS and Galileo Receiver* [*A Single-Frequency Approach*] Birkhäuser, 2007

[4] P. Kovář, *Družicová navigace.* [*Od teorie k aplikacím v softwarovém přijímači*] České vysoké učení technické v Praze, Česká technika - nakladatelství ČVUT, 2016.

[5] European Space Agency: Navipedia
https://gssc.esa.int/navipedia/