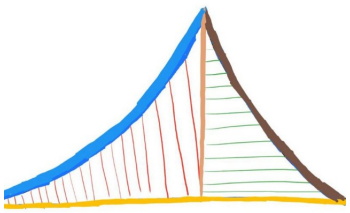
THE NAVIC STANDARD

Through Python

G. V. V. Sharma



Contents

Introduction

This book introduces the NAVIC communication standard through Python exercises

Chapter 1

Design Parameters

1.1. The Frequency Bands

An independent Indian Satellite based positioning system for critical National applications. The main objective is to provide Reliable Position, Navigation and Timing services over India and its neighbourhood, to provide fairly good accuracy to the user.

Table 1.1: the navic frequency bands

Signal	Carrier Frequency	Bandwidth	Usage
L1	1575.42 Mhz	24 Mhz	for low power devices
L5	1176.45 Mhz	24 Mhz	navigation and positioning
S	2492.028 Mhz	16.5 Mhz	SBAS and messaging

Because of satellites' increased use, number and size, congestion has become a serious issue in the lower frequency bands
The higher frequency bands1.1 typically give access to wider bandwidths, but are also more susceptible to signal degradation due to 'rain fade' (the absorption of radio signals by atmospheric rain, snow or ice).

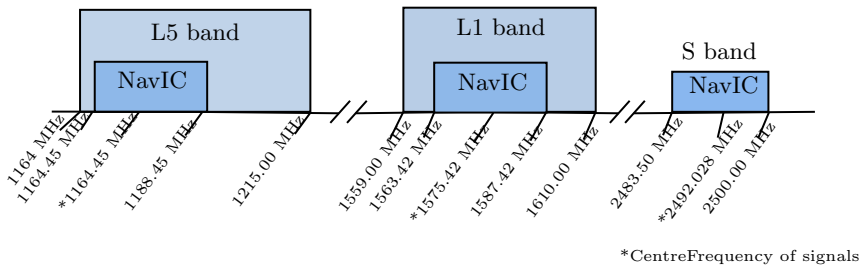


Figure 1.1: Frequency bands of NavIC Signals

1.1.1. L-band

NavIC carriers and also satellite mobile phones, such as Iridium; Inmarsat providing communications at sea, land and air; WorldSpace satellite radio.

1.1.2. S-band

Weather radar, surface ship radar, and some communications satellites, especially for communication with ISS and Space Shuttle.

With the variety of satellite frequency bands that can be used, designations have been developed so that they can be referred to easily.

1. Single frequency NavIC receiver capable of receiving signal in L1/L5/S band frequency.

2. A multi-frequency NavIC receiver capable of receiving combinations of L1, L5 and S band signals.

3. A multi-constellation receiver compatible with NavIC and other GNSS signals.

Each NavIC satellite provides SPS signals in L1, L5 and S bands.

1.2. NavIC carrier frequencies

The seven satellites in the NavIC constellation so far use two frequencies for providing positioning data — the L5 and S bands. The new satellites NVS-01 onwards, meant to replace these satellites, will also have L1 frequency.

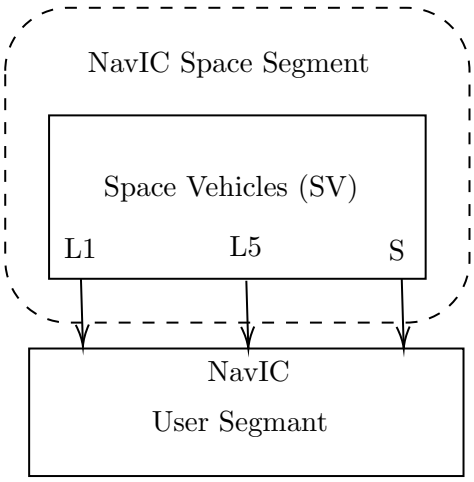


Figure 1.2: the navic bands segment blocks

The Figure1.2 above specifies the radio frequency interface between space and user segments.

The NavIC will provide basically two types of services:

1. Standard Positioning Service (SPS)
2. Restricted Service (RS)

1.2.1. Standard Positioning Service (SPS)

It is available to all civilian users free of charge and provides positioning, navigation, and timing information with a moderate level of accuracy. The

SPS signals in NavIC primarily operate in the L5 frequency band1.1.

1.2.2. Restricted Service (RS)

The RS is intended for authorized users and offers enhanced accuracy, integrity, and availability compared to the SPS signals. The RS signals in NavIC operate in both the L5 and S bands1.1.

Both services will be carried on L5 and S band1.1. The navigation signals would be transmitted in the S-band frequency and broadcast through a phased array antenna to keep required coverage and signal strength.

navic operated only in the L5-band and S-band frequencies. This was because India hadn't received the International Telecommunication Union authorisation for using the L1 and L2 frequency bands, which are widely used worldwide for navigation services.

Now that L1 band1.1 is available on the NVS-01 satellite(and will be available on subsequent NVS satellites), it is an interoperable frequency and can be used across all chipsets(of mobile devices), provided they use our signal architecture

All NavIC satellites transmit navigation signals in two or more frequency bands as in the table1.1. These signals contain ranging codes that allow receivers to compute their travelling time from satellite to receiver, along with navigation data, in order to know the satellite's position at any time.

Chapter 2

Transmitter

2.1. Frame structure

NavIC master frame consists of 2400 symbols, divided to four subframes. Each subframe is 600 symbols long. Subframes 1 and 2 transmit fixed navigation parameters. Subframe 3 and 4 transmit secondary navigation parameters in the form of messages. Each subframe is 292 bits long without FEC encoding and sync word. It starts with TLM word of 8 bits. Ends with 24 bit Cyclic Redundancy Check(CRC) followed by 6 tail bits. In subframes 1 and 2 navigation data is alloted 232 bits, starting from bit 31. In subframe 3 and 4, 220 bits are alloted starting from bit 37. For detailed structure of subframes, refer to chapter 5.9 in the doc

2.1.1. Cyclic Redundancy Check(CRC)

The parity coding of data signal follows 24Q polynomial for each subframe. 24 bits of CRC parity will provide protection against burst as well as random errors with undetected eroor probability of 2^{-24} for all channel bit error

probabilities 0.5

$$g(X) = \sum_{i=0}^{24} g_i X^i \;\; g_i = 1 \; for \; i = 0, 1, 3, 4, 5, 6, 7, 10, 11, 14, 17, 18, 23, 24 \tag{2.1}$$

2.2. Encoding

The navigation data subframe of 292 bits is rate 1/2 convolution encoded and clocked at 50 symbols per second. Each subframe of 292 bits after encoding results in 584 bits. For parameters and coding scheme, refer to below doc

2.2.1. Interleaving

Any burst errors during the data transmission can be corrected by interleaving. In matrix interleaving, input symbols are filled into a matrix column wise and read at the output row wise. This will spread the burst error, if any, during the transmission. For SPS, data is filled into matrix of size 73 by 8(73 columns, 8 rows).

2.2.2. Sync word and Tail bits

Each subframe has a 16 bit word synchronization pattern which is not encoded. Sync pattern is EB90 Hex. Tail bit consists of 6 zero value bits enabling completion of FEC decoding of each subframe in the receiver.

2.3. Modulation

2.3.1. Standard Positioning Service

The SPS signal is BPSK(1) modulated on L5 and S bands. The navigation data at data rate of 50 sps (1/2 rate FEC encoded) is modulo 2 added to PRN code chipped at 1.023 Mcps. The CDMA modulated code, modulates the L5 and S carriers at 1176.45 MHz and 2492.028 MHz respectively.

2.3.2. Pseudo Random Noise codes(PRN)

NavIC uses Gold codes fo SPS signal. They are generated using Linear Feedback Shift Registers. For L5 and S band, the code length is 1ms and consists of 1023 chips. The code is chipped at 1.023 Mcps. Two polynomials G1 and G2 are used to generate the gold code sequence. G2's initial state provides unique PRN code for each satellite. All bits of G1 are initialized as 1. G1 and G2 are XOR'ed to generate final 1023 chip long PRN sequence, the time period being 1ms. For more information refer to chapter 4 in the doc.

2.3.3. Baseband Modulation

The carrier signal is modulated by BPSK(1), Data channel BOC(5,2), and Pilot Channel BOC(5,2). To have a constant envelop when passed through power amplifier, we add additional signal called interplex signal.

2.3.3.1. Mathematical Equations

SPS Data Signal

$$s_{sps}(t) = \sum_{i=-\infty}^{\infty} c_{sps}(|i|_{L\_sps}).d_{sps}(\,[i]_{CD\_sps}).rect_{T_{c,sps}}(t - iT_{c,sps}) \tag{2.2}$$

RS BOC Pilot Signal

$$s_{rs\_p}(t) = \sum_{i=-\infty}^{\infty} c_{rs\_p}(|i|_{L\_rs\_p}).rect_{Tc,rs\_p}(t - iT_{c,rs\_p}).sc_{rs\_p}(t, 0) \tag{2.3}$$

RS BOC Signal

$$s_{rs\_d}(t) = \sum_{i=-\infty}^{\infty} c_{rs\_d}(|i|_{L\_rs\_d}).d_{rs\_d}(\,[i]_{CD\_rs\_d}).rect_{T_{c,rs\_d}}(t - iT_{c,rs\_d}).sc_{rs_d}(t, 0) \tag{2.4}$$

The sub-carrier is defined as:

$$sc_x(t, \phi) = sgn\,[sin(2\pi f_{sc,x}t + \phi)] \tag{2.5}$$

The IRNSS RS data and pilot BOC signals are sinBOC. Hence the sub-carrier phase $\phi = 0$. The complex envelope of composite signal with Interplex signal (I(t)) is:

$$s(t) = \frac{1}{3}\left[\sqrt{2}(s_{sps}(t) + s_{rs\_p}(t)) + j(2.s_{rs\_d}(t) – I(t))\right] \tag{2.6}$$

The Interplex signal $I(t)$ is generated to realize the constant envelope composite signal. The operation $|i|_X$ gives the code chip index for any signal. Similarly $[i]_X$ gives data bit index for any signal. Symbol definitions are given

in below table 2.2.

Symbol	Definition
A	received signal amplitude
f_c	carrier frequency
f_{sub}	subcarrier frequency
t	time
q	phase offset
s(t)	BPSK signal transmitted data(-1,1)

Table 2.2: Symbol Description

The functions for data generation and baseband modulation are present in the below code,

codes/transmitter/transmitter.py

Chapter 3

Channel Modelling

The phenomena modelled in the satellite communication channel are Doppler shift, delay, power scaling and thermal noise at the receiver.

3.1. Doppler shift

Due to relative motion between the satellites and the receiver, the transmitted signals undergo a frequency shift before arriving at the receiver. This shift in frequency is called Doppler shift and can be computed as

$$f_{shift} = f_d - f_c = \left(\frac{V_{rel}}{c - V_{S,dir}} \right) f_c \tag{3.1}$$

where,

f_{Shift} = Frequency shift due to Doppler

f_d = Frequency observed at receiver

f_c = Carrier frequency at transmitter

V_{rel} = Relative velocity of transmitter and receiver

$V_{S,dir}$ = Velocity of satellite along radial direction

c = Speed of light

V_{rel} is given by

$$V_{rel} = V_{S,dir} - V_{R,dir} \tag{3.2}$$

where,

$V_{R,dir}$ = Velocity of receiver along radial direction

$V_{R,dir}$ and $V_{S,dir}$ are given by

$$V_{R,dir} = \mathbf{V}_R \cdot \hat{\mathbf{dir}} \tag{3.3}$$

$$V_{D,dir} = \mathbf{V}_S \cdot \hat{\mathbf{dir}} \tag{3.4}$$

where,

$\hat{\mathbf{dir}}$ = Unit vector from satellite to receiver i.e. radial direction

$\mathbf{V_S}$ = Velocity of satellite

$\mathbf{V_R}$ = Velocity of receiver

$\hat{\mathbf{dir}}$ is given by

$$\hat{\mathbf{dir}} = \frac{\mathbf{P_S} - \mathbf{P_R}}{\|\mathbf{P_S} - \mathbf{P_R}\|} \tag{3.5}$$

where,

$\mathbf{P_S}$ = Position of satellite

$\mathbf{P_R}$ = Position of receiver

The Doppler shift is introduced by muliplying the satellite signal with a

complex exponential,

$$x_{Shift}[n] = x[n]\, e^{-2\pi j\left(f_c + f_{Shift}\right) n t_s} \tag{3.6}$$

where,

$x_{Shift}[n]$ = Doppler shifted signal

$x[n]$ = Satellite signal

t_s = Sampling period

3.2. Delay

Since there is a finite distance between the satellite and the receiver, the signal at the reciever is a delayed version of the transmitted signal. This delay is given by

$$D_s = \frac{d}{c} f_s \tag{3.7}$$

where,

D_s = Total delay in samples

d = Distance between satellite and receiver

c = Speed of light

f_s = Sampling rate

The total delay on the satellite signal is modeled in two steps. First, a static delay is modeled which does not change with time and it is always an integer number of samples. Then, a variable delay is modeled which can be a rational number of samples. While modelling the static delay, the

entire delay is not introduced so that variable delay modelling handles the remaining delay.

To introduce the static delay, the samples are read from a queue whose size is the desired static delay length. When samples are read from the queue, an equal number of new samples are updated in the queue. To introduce the variable delay, the signal is passed through an all-pass FIR filter with an almost constant phase response. Its coefficients are calculated using the delay value required.

3.3. Power Scaling

When a transmitting antenna transmits radio waves to a receiving antenna, the radio wave power received is given by,

$$P_r = P_t D_t D_r \left(\frac{1}{4\pi \left(f_c + f_{Shift} \right) D} \right)^2 \tag{3.8}$$

where,

P_r = Received power

P_t = Transmitted power

D_t = Directivity of transmitting antenna

D_r = Directivity of receiving antenna

D = Total delay in seconds

To scale the received signal as per the received power calculated,

$$x_{Scaled}\left[n\right] = \frac{\sqrt{P_r}}{\mathrm{rms}\left(x\left[n\right]\right)} x\left[n\right] \tag{3.9}$$

3.4. Thermal noise

The thermal noise power at the receiver is given by,

$$N_r = kTB \tag{3.10}$$

where,

N_r = Noise power in watts

k = Boltzmann's constant

T = Temperature in Kelvin

B = Bandwidth in Hz

AWGN (Additive White Guassian Noise) samples with zero mean and variance N_r are generated and added to the satellite signal to model thermal noise at receiver.

The functions necessary to model the channel are present in the below code,

codes/channelmodel/channelmodel.py

Chapter 4

Receiver

4.1. Hardware Implementation

List the various components used to implement receiver.

Solution:

Components are listed in the table Table 4.2

Component	Type
RTL-SDR chip	Hardware
GNU radio	Software
Antenna	Hardware

Table 4.2: Components Required

The picture of RTL-SDR and Antenna is given in Fig. 4.2.This set is used to receive the L5,L1,S Band signals.

4.2. RTL SDR Specifications

RTL-SDR is a popular, low cost hardware that can receive wireless signals. The RTL-SDR dongle features the Realtek RTL2838U chip, which provides

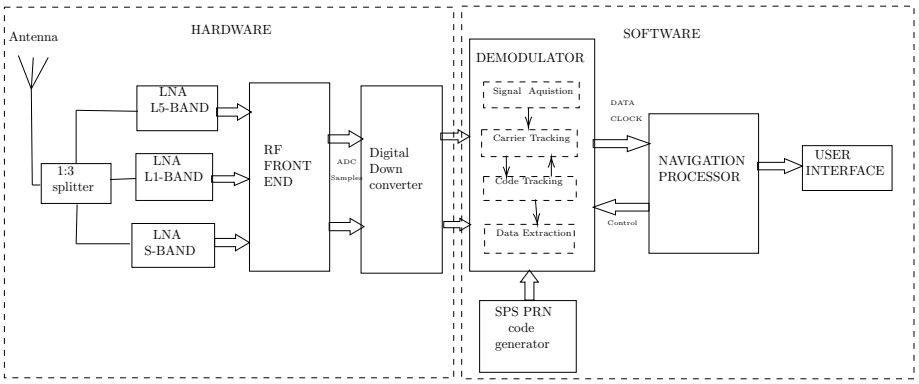


Figure 4.1: The Block Level Architecture

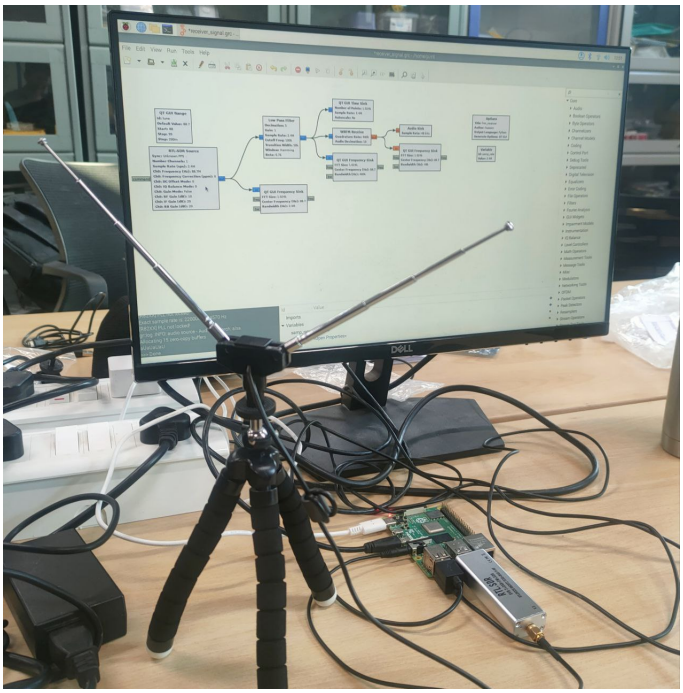


Figure 4.2: Reciever Hardware implementation

Figure 4.3: RTL-SDR

I-Q samples through the USB interface.

Name	**RTL-SDR(R820T2/RTL2838U DVB-T)**
Type	Pre-built and premodded with custom driver
Frequency range	0.5-1766MHz
Max Bandwidth	Matches sampling rate but with filter roll-off
Receiver ADC bits	8
Tx.DAC bits	-
Tx. Cable	No
Sampling Rate	2.4 MHz
Frequency accuracy	1 ppm
Host inerface	USB
FPGA	-

Table 4.4: RTL-SDR Specification table

Install and open GNU Radio using the following commands

```
sudo apt update
sudo apt install gnuradio
gnuradio−companion
```

How to construct the block diagram in GNU radio?

Solution:

Step-1:

Search for QTGUI Time sink block and add it to the work space. **Step-2**:

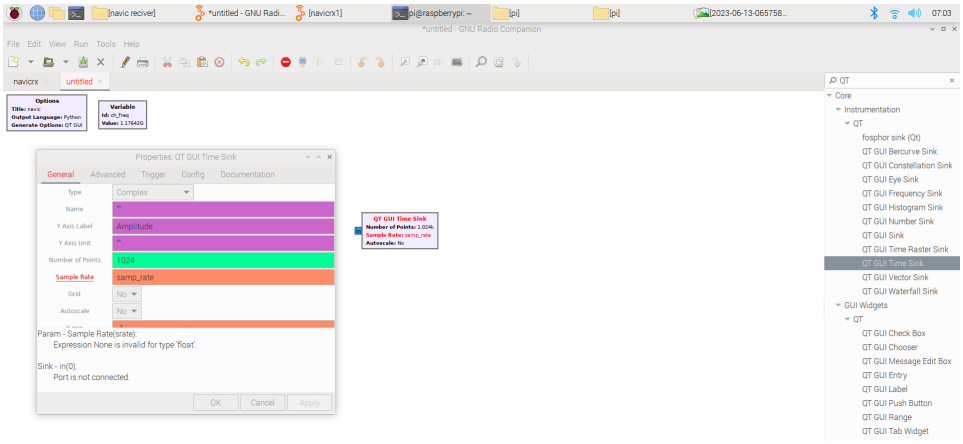


Figure 4.4: Adding blocks

Similarly do for RTL-Source block,complex to real ,complex to imaginary and QTGUI Time sink block.

Step-3: Change the parameters in each block according your values by double clicking on it.

Step-4: Connect them according to the flowgraph shown in Fig. 4.5. **Note**: Refer the following website for any queries.

https://wiki.gnuradio.org/index.php?title=Creating_Your_First_Block

Explain each block in block diagram of Receiver.

Solution:

1. RTL-SDR Source:

The RTL-SDR Source block is used to stream samples from RTL-SDR de-

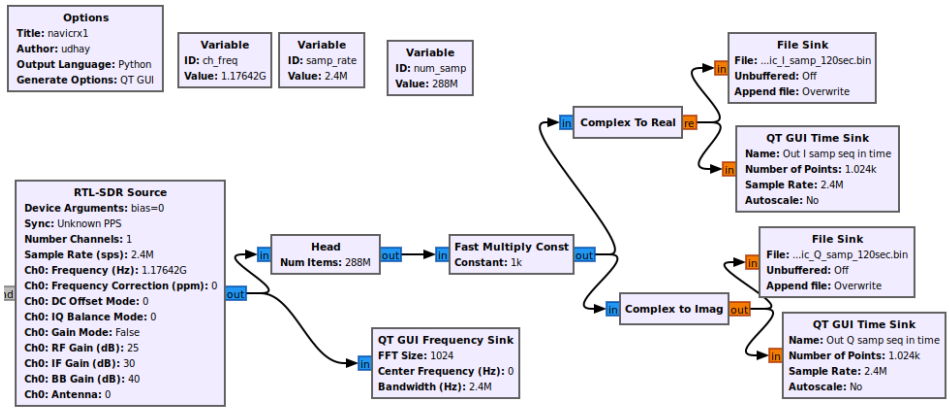


Figure 4.5: Block diagram of Receiver in GNU Radio

vice. Connect the RTL-SDR to the system and execute the flowgraph in

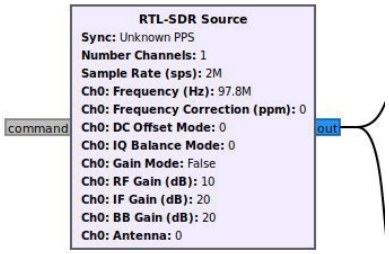


Figure 4.6: RTL-SDR Source Block

Fig. 4.5.

4.2.1. Results

I,Q samples will be stored in a file with .bin extension

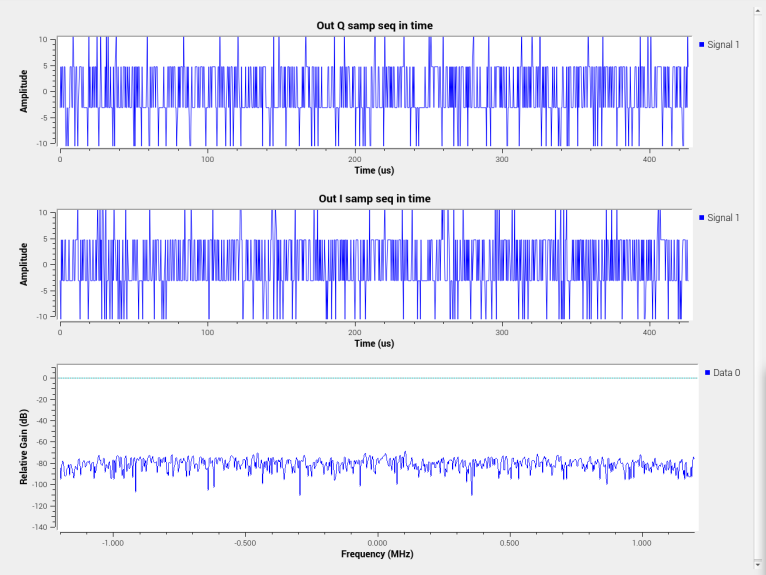


Figure 4.7: RTL-SDR receiver plots

Read the data from RTL SDR stored in .bin file using python code and without using GNU radio.

Solution:

The following code reads the data from RTL-SDR without GNU radio.

codes/receiver/rtsdr_rx.py

4.3. USRP SDR

The picture of Universal software Radio Peripheral-software defiend radios (USRP-SDR) is given in Fig. 4.8.This set is used to receive the L5,L1,S Band signals.

Figure 4.8: USRP-SDR

4.3.1. USRP SDR specification

Name	USRP B210
Type	Pre-built
Frequency range	70MHz-6GHz
Max Bandwidth	56MHz
Receiver ADC bits	12
Tx.DAC bits	12
Tx. Cable	Yes
Sampling Rate	50 Msps
Frequency accuracy	-
Host inerface	USB 3.0
FPGA	Xilinx Spartan 6 XC6SLX150

Table 4.6: USRP-SDR Specification table

Install and open GNU Radio using the following commands

```
sudo apt update
sudo apt install gnuradio
gnuradio−companion
```

How to construct the block diagram in GNU radio?

Solution:

Step-1:

Search for QTGUI Time sink block and add it to the work space.

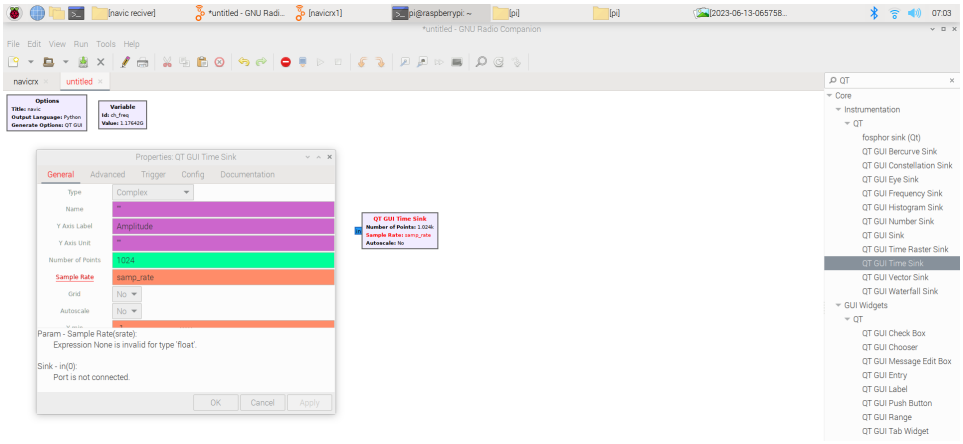


Figure 4.9: Adding blocks

Step-2: Similarly do for USRP-Source block,complex to real ,complex to imaginary and QTGUI Time sink block.

Step-3: Change the parameters in each block according your values by double clicking on it.

Step-4: Connect them according to the flowgraph shown in Fig. 4.5.

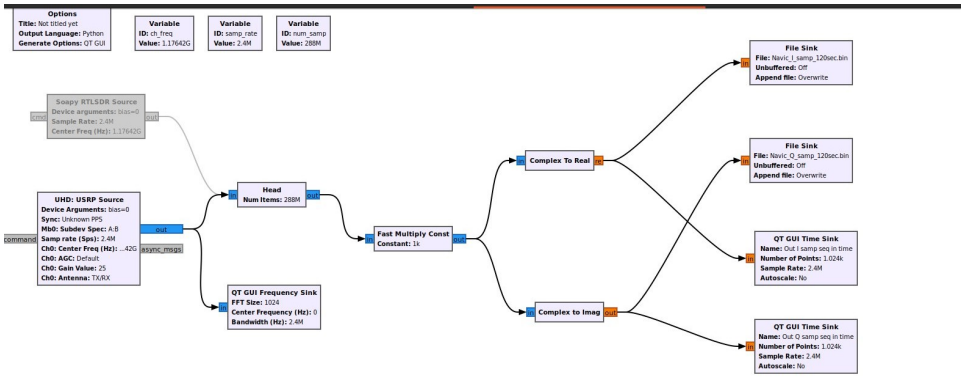


Figure 4.10: Block diagram of Receiver in GNU Radio

Note: Refer the following website for any queries.

https://wiki.gnuradio.org/index.php?title=Creating_Your_First_Block

Explain each block in block diagram of Receiver.

Solution:

1. RTL-SDR Source:

The RTL-SDR Source block is used to stream samples from USRP-SDR device.

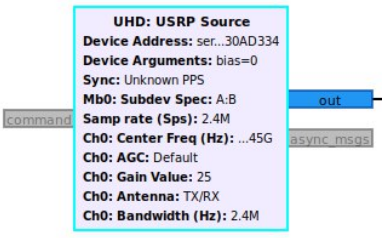


Figure 4.11: USRP-SDR Source Block

Connect the USRP-SDR as shown flowgraph in Fig. 4.10.

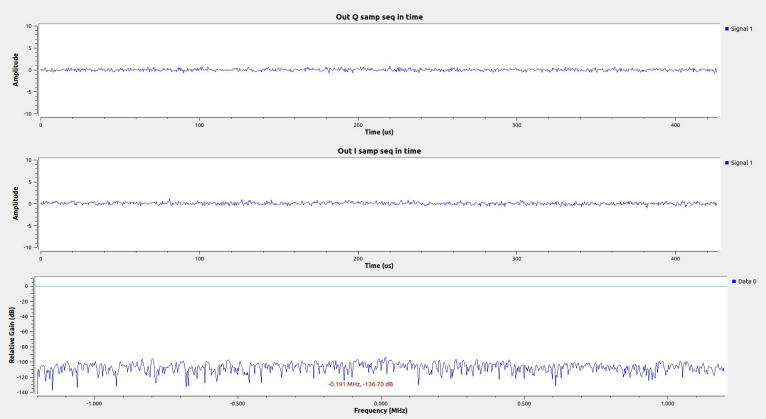


Figure 4.12: USRP receiver plots

4.3.2. Results

I,Q samples will be stored in a file with .bin extension

Read the data from RTL SDR stored in .bin file using python code and without using GNU radio.

Solution:

The following code reads the data from RTL-SDR without GNU radio.

codes/receiver/usrp_rx.py

Chapter 5

Demodulation

5.1. what is Demodulation

Demodulation is the process of extracting the original information or baseband signal from a modulated carrier signal.The purpose of demodulation is to retrieve the modulating signal, which could be analog or digital data, audio, video, or other forms of information. Demodulation is essential in various communication systems such as radio, television, cellular networks, and wireless data transmission.

The demodulation code used in NavIC may vary depending on the specific implementation and receiver hardware. However, I can provide you with a general outline of the demodulation process for NavIC signals.

NavIC signals are transmitted in the L5 frequency band (1176.45 MHz) using BPSK modulation for the navigation message and various modulation schemes (such as BOC and MBOC) for the ranging signal. The demodulation process typically involves the following steps:

Signal Acquisition: The receiver searches for and acquires the NavIC signal by correlating the received signal with a locally generated replica of

the spreading code used by the satellites. This process helps in identifying the presence of the NavIC signal and estimating the initial timing offset.

Carrier Tracking: Once the signal is acquired, the receiver performs carrier tracking to estimate and track the carrier frequency and phase of the received signal. This is crucial for demodulation as it ensures accurate demodulation of the navigation message and ranging signal.

Code Tracking: The receiver performs code tracking to estimate and track the spreading code used by the satellites. This helps in maintaining synchronization with the transmitted signal and extracting the navigation data and ranging information.

Data Extraction: Once the demodulation is performed, the receiver extracts the navigation data from the demodulated signal. The navigation data includes information such as satellite ephemeris, clock correction, and other parameters necessary for calculating the receiver's position, velocity, and timing.

5.1.1. Signal Acquistion

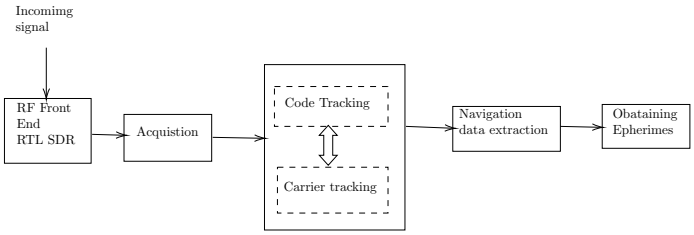


Figure2 : The Block Level Architecture in Demodulation

A generic IRNSS signal defined by its complex baseband equivalent, $S_T(t)$, the digital signal at the input of an Acquisition block can be written as:

$$x_{IN}[k] = A(t)\hat{S}_T(t-\tau(t))e^{j(2\pi f_D(t)t+\Phi))}\Big|_{t=kT_s} + n(t)\Big|_{t=kT_s} \tag{5.1}$$

Table 5.1: Parameters Table in Signal Acquisition

Symbol	Definition
$x_{IN}[k]$	complex vector I, Q samples of received signal
A(t)	Signal Amplitude
$\hat{S}_T(t)$	filtered version of $S_T(t)$
$f_D(t)$	Time - varing doppler shift
$\Phi(t)$	Time - varing carrier phase shift
n(t)	Term modelling random noise
T_s	sampling period

5.1.1.1. Implementation of CA PCPS Acquisition

The Parallel Code Phase Search (PCPS) algorithm is described as follows: Input signal buffer x_{IN} of k complex samples, acquisition threshold γ provided by the Signal Conditioner; on-memory FFT of the local replica,

$$D[k] = FFT_k\{d[k]\} \tag{5.2}$$

$$\text{input signal power estimation :} \hat{P}_{in} = \frac{1}{k}\sum_{k=0}^{k-1}\left|x_{IN}[k]\right|^2 \tag{5.3}$$

$$\check{f}_D = [f_{min}, f_{max}] \quad in \quad f_{steps} \tag{5.4}$$

carrier wipe off: $x[k] = x_{IN}[k]e^{-(j2\pi\check{f}_D kT_s)}, \quad for \quad k = 0, ..., k-1$ (5.5)

$$X[k] = FFT_k\{x[k]\} \tag{5.6}$$

$$Y[k] = X[k].D[k], \quad fork = 0, ...k-1. \tag{5.7}$$

$$R_{xd}(\check{f}_D, \tau) = \frac{1}{k^2} IFFT_k\{Y[k]\} \tag{5.8}$$

Search maximum and its indices in the search grid:

$$\{S_{max}, f_i, \tau_j\} = max_{f,\tau}\left|R_{xd}(f, \tau)\right|^2 \tag{5.9}$$

Compute the Generalized Likelihood Ratio Test (GLRT) function with normalized variance:

$$\Gamma_{GLRT} = \frac{2kS_{max}}{\hat{P}_{in}} \tag{5.10}$$

if $\Gamma_{GLRT} > \gamma$

Declare positive acquisition and provides coarse estimation of code phase

$\hat{\tau}_{acq} = \tau_j$and Doppler shift $\hat{f}_{D_{acq}} = f_i$,

other wise declare negative acquisition.

The acquisition results are generated using the below code

Code

code/e2e_sim/main.ipynb

Result

Acquisition results for PRN ID 5
Status:True Doppler:3500 Delay/Code−Phase:300/30.0
Acquisition results for PRN ID 7
Status:True Doppler:2500 Delay/Code−Phase:587/58.7
Acquisition results for PRN ID 3
Status:True Doppler:1500 Delay/Code−Phase:426/42.6
Acquisition results for PRN ID 1
Status:True Doppler:2500 Delay/Code−Phase:313/31.3

5.1.2. Carrier tracking loop:

Phase locked loop(PLL)

The carrier loop discriminator defines the type of tracking loop as a PLL, aCostas PLL (which is a PLL-type discriminator that tolerates the presence of data modulation on the baseband signal), or a frequency lock loop (FLL).Carrier tracking loop tracks the frequency and phase of the received signal by detecting the phase error between replicated signal and incoming signal and accordingly replicated signal produced by numerically controlled

oscillator (NCO) is adjusted to synchronize with incoming signal in both frequency and phase. For zero phase error detected, navigation data is accurately extracted.

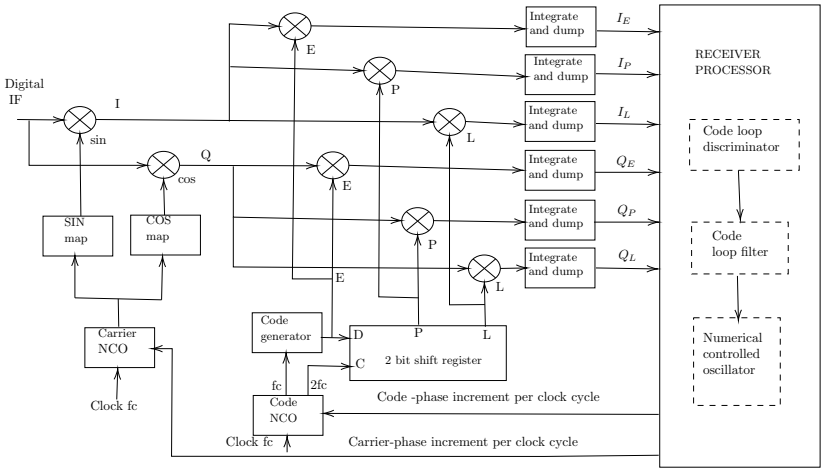


Figure3 : Generic digital receiver channel

$$\text{PLL Discriminator} = ATAN2(I_P, Q_P) = \tan^{-1}\left(\frac{I_P}{Q_P}\right) \tag{5.11}$$

compares the phase error outputs of these PLL discriminators assuming no noise in the I and Q signals. Note that the ATAN2 discriminator is the only one that remains linear over the full input error range of $\pm 180°$. However, in the presence of noise, both of the discriminator outputs are linear only near the $0°$ region. These PLL discriminators will achieve the 6-dB improvement in signal tracking threshold (by comparison with the Costas discriminators described next) for the dataless carrier because they track the full four quadrant range of the input signal.

Frequency locked loop

PLLs replicate the exact phase and frequency of the incoming SV (converted to IF) to perform the carrier wipeoff function. FLLs perform the carrier wipeoff process by replicating the approximate frequency, and they typically permit the phase to rotate with respect to the incoming carrier signal.

$$\text{FLL Discriminator} = \frac{ATAN2(dot, cross)}{t2 - t1} \tag{5.12}$$

ATAN2(dot, cross)FLL discriminators are equal to half the predetection bandwidths. The (cross)× sign(dot) FLL discriminator frequency pull-in ranges are only one-fourth of the predetection bandwidths. Also note that the (cross) × sign(dot) and the cross FLL discriminator outputs, whose outputs are sine functions divided by the sample time interval $(t_2 - t_1)$ in seconds, are also divided by 4 to more accurately approximate the true input frequency error. The ATAN2 (x, y) function returns the answer in radians, is converted to degrees, divided by the sample time interval $(t_2 - t_1)$in seconds, and is also divided by 360 to produce at its output a true representation of the input frequency error within its pull-in range

5.1.3. Code tracking loop:

Delay locked loop Post the carrier signal synchronization, received CA code samples is synchronized by aligning with replicated CA code samples by shifting right or left. To determine the direction of shift, the I and Q outputs are multiplied with prompt code (PRN code which is phase aligned),

early code (prompt PRN code shifted by some samples to the right) and late code (prompt PRN code shifted by some samples to the left) resulting in corresponding to I and Q channel respectively.IRNSS receiver delay lock loop (DLL) discriminators and their characteristics.

$$E = \sqrt{I_{ES}^2 + Q_{ES}^2} \tag{5.13}$$

$$L = \sqrt{I_{LS}^2 + Q_{LS}^2} \tag{5.14}$$

$$\text{DLL Discriminator}(\epsilon) = \frac{1}{2}\frac{E-L}{E+L} \tag{5.15}$$

Noncoherent early minus late envelope normalized by $E+L$ to remove amplitude sensitivity High computational load.For 1-chip $E-L$ correlator spacing, produces true tracking error within ± 0.5 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.

5.1.4. Loop filter characteristics

The values for the second-order coefficient a_2 and third-order coefficients a_3 and b_3 can be determined from Table 3. These coefficients are the same for FLL, PLL, or DLL applications if the loop order and the noise bandwidth,B_n , are the same.Note that the FLL coefficient insertion point into the filter is

Table 5.3: Loop order filters

Loop Order	**Noise Bandwidth B_n (Hz)**	**Typical Filter Values**
First	$\frac{\omega_o}{4}$	ω_o $B_n = 0.25\omega_o$
Second	$\frac{\omega(1+a_2^2)}{4a_2}$	ω_o^2 $a_2\omega_o = 1.414\omega_o$ $B_n = 0.53\omega_o$
Third	$\frac{\omega(a_3b_3^2+a_3^2-b_3)}{4(a_3b_3-1)}$	ω_o^3 $a_3\omega_o^2 = 1.1\omega_o^2$ $b_3\omega_o = 2.4\omega_o$ $B_n = 0.7845\omega_o$

one integrator back from the PLL and DLL insertion points.This is because the FLL error is in units of hertz (change in range per unit of time).

The tracking results are generated using the below code

Code

code/e2e_sim/main_ipynb

Result

Tracking result for PRN ID:5

Transmitted Bits:

[0. 1. 0. 1. 0. 0. 0. 1. 0. 1. 1. 0. 0. 0. 1. 1. 1. 0. 0. 1. 1. 1. 1. 0.1. 0. 1. 1. 0.
1. 0. 0. 1. 0. 0. 0. 1. 0. 1. 1. 1. 0. 1. 1. 0. 0. 1. 0. 0. 0.]

Received bits:

[1. 1. 0. 1. 0. 1. 1. 1. 0. 1. 0. 0. 1. 1. 1. 0. 0. 0. 1. 1. 0. 0. 0. 0.1. 0. 1. 0. 0.
1. 0. 1. 1. 0. 1. 1. 1. 0. 1. 0. 0. 0. 1. 0. 0. 1. 1. 0.1. 1.]

Received bits inverted:

[0. 0. 1. 0. 1. 0. 0. 0. 1. 0. 1. 1. 0. 0. 0. 1. 1. 1. 0. 0. 1. 1. 1. 1.0. 1. 0. 1. 1.
0. 1. 0. 0. 1. 0. 0. 0. 1. 0. 1. 1. 1. 0. 1. 1. 0. 0. 1. 0. 0.]

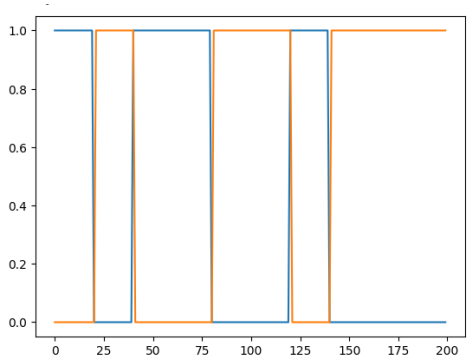


Figure 5.1: tracking result plot

The functions for acquisition and tracking are present in the below code

codes/demodulation/demodulation.py

Chapter 6

Channel Decoding

6.1. Definition

Channel decoding in NAVIC, the Indian Regional Navigation Satellite System, involves the process of error correction and retrieval of the original data transmitted over the satellite link. The channel decoding scheme used in NAVIC is based on a convolutional coding technique known as Rate 1/2 Convolutional Code with Viterbi decoding.

6.1.1. Process

Here is a high-level description of the channel decoding process in NAVIC:

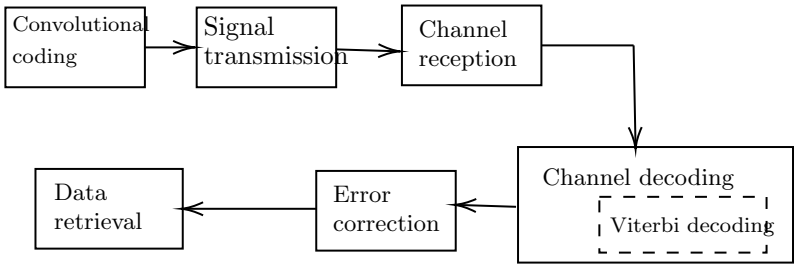


Figure 6.1: The Block Level Architecture in Channel Decoding

6.1.2. Convolutional Code Representation:

The convolutional code is represented as a state diagram or trellis, where each state represents a unique history of the encoded bits. The trellis consists of nodes and branches. Nodes correspond to states, and branches represent transitions between states. Each branch is labeled with the input bit and the encoded output bits associated with the transition.

6.1.3. Branch Metrics:

At each time step, the Viterbi algorithm calculates branch metrics, which quantify the similarity between the received signal and the expected signal for each branch. The branch metric is typically based on a distance measure, such as Hamming distance or Euclidean distance, between the received signal and the expected signal. Let's denote the received signal at time step t as r(t) and the expected signal for a particular branch as c(t). The branch metric B(t) for that branch at time step t is computed as the distance between r(t) and c(t).

6.1.4. Path Metrics:

The Viterbi algorithm computes a path metric for each state at each time step, which represents the accumulated likelihood of reaching that state along a particular path. The path metric is typically computed as the minimum (or maximum, depending on the metric used) of the sum of the previous path metric and the branch metric. Let's denote the path metric for

state i at time step t as P(i, t). The path metric for state i at time step t is computed as:

P(i, t) = minP(j, t-1) + B(t), where j is the previous state connected to state i.

6.1.5. Survivor Paths:

Along with the path metrics, the Viterbi algorithm keeps track of survivor paths, which represent the most likely paths leading to each state at each time step. The survivor paths are determined based on the branch with the smallest (or largest, depending on the metric used) branch metric leading to each state. The survivor paths help in traceback, as they indicate the most likely sequence of states leading to the current state.

6.1.6. Traceback:

Once the decoding reaches the end of the received signal, a traceback process is performed to determine the final decoded sequence. Starting from the state with the highest path metric at the last time step, the algorithm traces back through the trellis by following the survivor paths. The traceback process continues until reaching the starting state at the first time step, yielding the decoded sequence of transmitted bits.

6.1.7. Decoding Output:

The traceback process generates the final decoded output, which should ideally match the original transmitted data. The decoded output undergoes error correction, such as using error-correcting codes like Reed-Solomon, to further enhance the reliability of the decoded sequence.

The Viterbi algorithm is an iterative process that calculates and updates the path metrics and survivor paths at each time step. It efficiently explores all possible paths through the trellis and selects the most likely path. This results in the recovery of the transmitted data even in the presence of noise and errors.

Let's solve the above mentioned step by step.

$$Received\ signal : r = [0, 1, 1, 0, 0, 1, 1] \tag{6.1}$$

$$Generator\ polynomials : G1 = 1 + D^2 + D^3 = 1 + D^2 + D^3 \tag{6.2}$$

$$G2 = 1 + D + D^3 = 1 + D + D^3 \tag{6.3}$$

We'll perform the Viterbi decoding process and calculate the path metrics and survivor paths.

Initialize path metrics and survivor paths:

$$P[0, 0] = P[1, 0] = 0 \tag{6.4}$$

$$S[0, 0] = S[1, 0] = [] \tag{6.5}$$

Calculate branch metrics and update path metrics and survivor paths at each time step:

Time step t = 1:

$$B[0,1] = Hamming\ distance(r[1], 00) = 1 \tag{6.6}$$

$$B[1,1] = Hamming\ distance(r[1], 11) = 2 \tag{6.7}$$

$$P[0,1] = min(P[0,0] + B[0,1], P[1,0] + B[1,1]) = min(0+1, 0+2) = 1 \tag{6.8}$$

$$P[1,1] = min(P[0,0] + B[0,1], P[1,0] + B[1,1]) = min(0+1, 0+2) = 1 \tag{6.9}$$

$$S[0,1] = S[i_{min}, 0] + (0 or 1) = S[0,0] + 0 = [0] \tag{6.10}$$

$$S[1,1] = S[i_{min}, 0] + (0 or 1) = S[0,0] + 0 = [0] \tag{6.11}$$

Time step t = 2:

$$B[0,2] = Hamming\ distance(r[2], 00) = 1 \tag{6.12}$$

$$B[1,2] = Hamming\ distance(r[2], 11) = 2 \tag{6.13}$$

$$P[0,2] = min(P[0,1] + B[0,2], P[1,1] + B[1,2]) = min(1+1, 1+2) = 2 \tag{6.14}$$

$$P[1,2] = min(P[0,1] + B[0,2], P[1,1] + B[1,2]) = min(1+1, 1+2) = 2 \tag{6.15}$$

$$S[0,2] = S[i_{min}, 1] + (0 or 1) = S[0,1] + 0 = [0,0] \tag{6.16}$$

$$S[1,2] = S[i_{min}, 1] + (0 or 1) = S[0,1] + 0 = [0,0] \tag{6.17}$$

Time step t = 3 (continued):

$$B[0,3] = Hamming\ distance(r[3], 00) = 1 \tag{6.18}$$

$$B[1,3] = Hamming\ distance(r[3], 11) = 2 \tag{6.19}$$

$$P[0,3] = min(P[0,2] + B[0,3], P[1,2] + B[1,3]) = min(2+1, 2+2) = 3 \tag{6.20}$$

$$P[1,3] = min(P[0,2] + B[0,3], P[1,2] + B[1,3]) = min(2+1, 2+2) = 3 \tag{6.21}$$

$$S[0,3] = S[i_{min}, 2] + (0or1) = S[0,2] + 0 = [0,0,0] \tag{6.22}$$

$$S[1,3] = S[i_{min}, 2] + (0or1) = S[0,2] + 0 = [0,0,0] \tag{6.23}$$

Time steps t = 4, 5, 6 (similar calculations as above).

Perform traceback:

Identify i_{max} = argmax(P[i, N-1]) = argmax(P[i, 6]) = argmax(P[0, 6], P[1, 6])

In this case,

i_{max} = 0 (as P[0, 6] = 5 and P[1, 6] = 4)

Start traceback from i_{max} = 0, following the survivor paths backward.

Traceback:

$$t = 6 : S[0, 6] = [0, 0, 0] \tag{6.24}$$

$$t = 5 : S[0, 5] = [0, 0] \tag{6.25}$$

$$t = 4 : S[0, 4] = [0] \tag{6.26}$$

$$t = 3 : S[0, 3] = [0, 0, 0] \tag{6.27}$$

$$t = 2 : S[0, 2] = [0, 0] \tag{6.28}$$

$$t = 1 : S[0, 1] = [0] \tag{6.29}$$

$$t = 0 : S[0, 0] = [] \tag{6.30}$$

The traceback process yields the decoded sequence: [0, 0, 0, 0, 0, 0, 0].

Therefore, the Viterbi decoding of the received signal [0, 1, 1, 0, 0, 1, 1] with the given generator polynomials G1 and G2 yields the decoded sequence [0, 0, 0, 0, 0, 0, 0].

6.2. Software

Below python code realizes the above construction : codes/decoder/decode.py

Appendix A

Computing the Position and Velocity of the satellite from the RINEX file

A.1. Installations

```
pip3 install pymap3d
pip3 install georinex
pip3 install itertools
pip3 install argparse
```

A.2. Algorithm

Algorithm for finding the position and velocity of satellite From Rinex file

1. Get the rinex file for NavIC satellite from the official website.

2. The Rinex file contains the observational file and navigation file.

3. Convert the Rinex file to CSV file using the python

   The below python function will convert the NavIC RINEX file to CSV file.

   codes/sat_pos_vel/rinex_to_csv/funcs.py

4. Remove the empty rows in csv file.The python function for removing empty rows is

   codes/sat_pos_vel/rinex_to_csv/funcs.py

5. Convert the csv file to list in python so that each row is corresponds to the parameters of the satellite. Function for converting the csv file to list is given as :

   codes/sat_pos_vel/rinexread/funcs.py

6. Process the above list with the formulas mentioned in chaprter 3

   The python function for finding the position of satellite is given as :

   codes/sat_pos_vel/position/funcs.py

7. The velocity of the satellte is computed by the function

   codes/sat_pos_vel/velocity/funcs.py

8. The distance between the satellite and receiver is obtained by the python package called **pymap3d**,using this package convert ECEF

to spherical coordinate frame.So that we obtain the distance between satellite and receiver.

9. These position and velocity of the satellite is used for computing the doppler shift.

The above alogorith will work for both GPS and NavIC satellite.If there is a problem in converting Navic RINEX file to csv file then follow the instructions below:

1. go to the mentioned folder in your laptop.

```
./home/username/.local/lib/python3.10/site-packages/georinex/
    nav3.py
```

2. Go to the line 220 in nav3.py file and modify the below changes.

```
elif numval == 29: # only one trailing spare fields
            cf = cf[:-2]
        elif numval == 28: # only one trailing spare fields
            cf = cf[:-3]
        elif numval == 27: # only one trailing spare fields
            cf = cf[:-4]
        elif numval == 26: # only one trailing spare fields
            cf = cf[:-5]
```