**IMPLEMENTATION OF ACQUISITION AND TRACKING ALGORITHMS OF GLONASS SOFTWARE RECEIVER IN C**

A

Dissertation Submitted in the partial fulfillment of the academic requirements for the award of degree of

**Bachelor of Engineering**

In

**ELECTRONICS and COMMUNICATION ENGINEERING**

By,

A JUDE DANIEL SUNIL (100510735016)

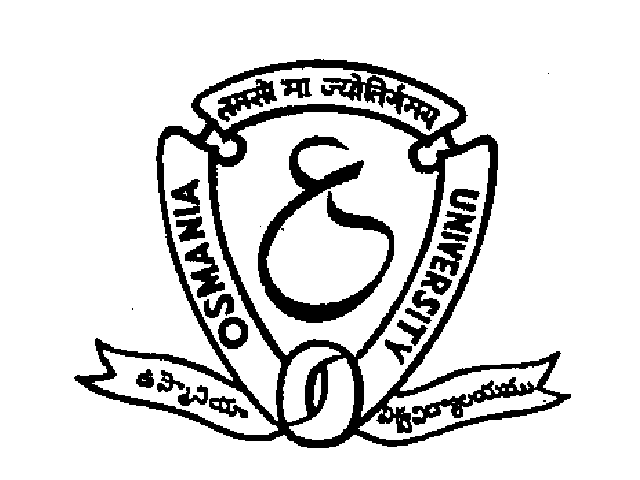
M.PRAGNA (100510735029)

K.SRAVYA (100510735039)

M.SWATHI (100510735043)

**Under the guidance of**

Dr.P.Laxminarayana, Principal Scientist (NERTU)

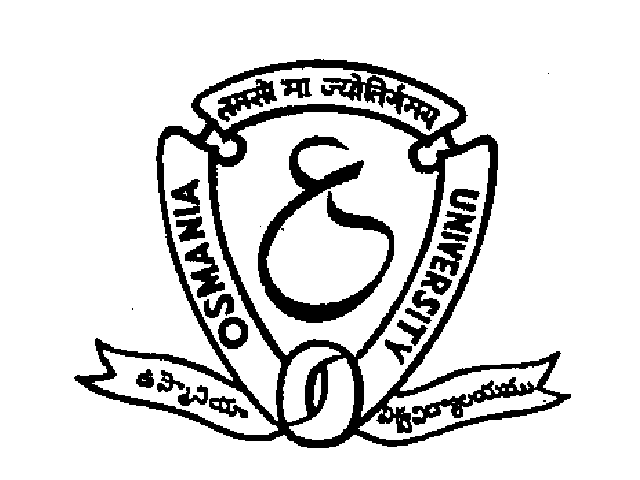


**Department of Electronics and Communications Engineering**

**University College of Engineering (Autonomous)**

**Osmania University, Hyderabad-500007**

**2013-2014**

****

**Department of Electronics and Communications Engineering**

**University College of Engineering (Autonomous)**

**Osmania University, Hyderabad-500007**

**2013-2014**

***Certificate***

This is to certify that this project work entitled “***IMPLEMENTATION OF ACQUISITION AND TRACKING ALGORITHMS OF GLONASS SOFTWARE RECEIVER IN C***” is a bonafide work carried out by A Jude Daniel Sunil (100510735016), M.Pragna (100510735029), K.Sravya (100510735039), M.Swathi (100510735043) in the partial fulfillment of the requirements for the award of “**Bachelor of Engineering**” degree in “**Electronics and Communication Engineering**” from “**University** **College of Engineering, Osmania University**”, Hyderabad, during the period 2013-2014 under our guidance and supervision.

**Project Guide Head of the Department**

Dr. P.Laxminarayana Dr.P.ChandraSekhar

(Principal scientist NERTU) (Dept of ECE, UCEOU)

**DECLARATION**

We hereby declare that the results in this dissertation work entitled **“IMPLEMENTATION OF ACQUISITION AND TRACKING ALGORITHMS OF GLONASS SOFTWARE RECEIVER IN C”** is the bonafide work done and carried out by us during the year 2013-14 in partial fulfillment of the academic requirements for the award of **Bachelor of Engineering** in Electronics and Communication Engineeringfrom **University College of Engineering**, **OSMANIA UNIVERSITY,** Hyderabad.

Further, we declare that the report has not been submitted by us to any other institute or university for award of any other degree.

**A Jude Daniel Sunil M.Pragna K.Sravya M.Swathi**

**(100510735016) (100510735029) (100510735039) (100510735043)**

# ACKNOWLEDGEMENT

The completion of the project gives us an opportunity to convey my gratitude to all those who have helped us to reach a stage where we have the confidence to launch our career in the competitive world of communication systems.

We are extremely thankful to our project guide **Dr. P Laxminarayana**, Principal Scientist, Research and Training Unit for Navigational Electronics (NERTU), Osmania University, Hyderabad for his excellent guidance and constant encouragement throughout the project work. It was a pleasant and challenging experience to work under his supervision. His constant monitoring and the valuable advices ensured the completion of this report. We thank him for setting high standards and motivating us to do the best.

We also thank **Dr.P.Chandra Sekhar**, Head, Department of Electronics and Communication Engineering, **Dr.R.Hemalatha**, Assistant Professor, Department of Electronics and Communication Engineering and **Dr.D.Ramakrishna**, Assistant Professor, Department of Electronics and Communication Engineering for their support from the Department. We would like to express our sincere thanks to **Director**, **NERTU**, **OU** and **NERTU Staff** and each and every person in the NERTU organization.

**A Jude Daniel Sunil M.Pragna K.Sravya M.Swathi**

**(100510735016) (100510735029) (100510735039) (100510735043)**

# ABSTRACT

These days, Global Navigation Satellite System (GNSS) technology plays a critical role in positioning and navigation applications. Use of GNSS is becoming more of a need to the public. Therefore, much effort is needed to make the civilian part of the system more accurate, reliable and available, especially for safety-of-life purposes. With the recent development of Russian Global Navigation Satellite System (GLONASS), it is worthwhile to concentrate on the GLONASS system as an alternative or an augmentation to GPS to achieve more reliable and accurate navigation solutions.

The use of GLONASS, in addition to GPS, provide very significant advantages such as increased satellite signal observation, and reduced horizontal and vertical dilution of precision factors. In particular, because of the limited number of GPS satellites, urban canyons and other locations with restricted visibility, such as forested areas, have problems acquiring and tracking a sufficient number of GPS signals. However, with the availability of combined GPS and GLONASS receivers, users will eventually have access to a 48-satellite constellation and the performance in these types of areas will improve accordingly. At present lot of research work on GPS software receiver is going and very few people are working on GLONASS software receiver. The basic difference in GPS and GLONASS in signal structure is CDMA and FDMA technologies.

The aim of this work is to implement acquisition and tracking algorithms which are the two main parts of Time synchronous GLONASS software receiver in C using Microsoft visual studio.

**Tools used:** Matlab, Microsoft Visual Studio 2010.

# CONTENTS

[ACKNOWLEDGEMENT 1](#_Toc387154407)

[ABSTRACT 2](#_Toc387154408)

[CONTENTS 3](#_Toc387154409)

[List of Figures 6](#_Toc387154410)

[List of Tables 8](#_Toc387154411)

[List of Abbreviations 9](#_Toc387154412)

[1. Introduction 10](#_Toc387154413)

[1.1 GNSS 10](#_Toc387154414)

[1.2 GNSS Software Receivers 11](#_Toc387154415)

[1.3 GLONASS Software Receiver 12](#_Toc387154416)

[1.4 Motivation 13](#_Toc387154417)

[1.5 Objective of the Project 14](#_Toc387154418)

[1.6 Organization of the Report 14](#_Toc387154419)

[2. GLONASS Architecture and Signal Structure 15](#_Toc387154420)

[2.1 Introduction 15](#_Toc387154421)

[2.2 Principle of Operation 15](#_Toc387154422)

[2.3 Architecture of GLONASS 16](#_Toc387154423)

[2.4 GLONASS Signal Structure 19](#_Toc387154424)

[2.4.1 Precision Code (P - code) 20](#_Toc387154425)

[2.4.2 Coarse Acquisition Code (C/A code) 21](#_Toc387154426)

[2.4.3 Frequency plan 23](#_Toc387154427)

[2.5 Correlation Properties of C/A Code 25](#_Toc387154428)

[2.6 Conclusion 27](#_Toc387154429)

[3. GNSS Software Receiver 28](#_Toc387154430)

[3.1 General overview of GNSS Receiver 28](#_Toc387154431)

[3.2 Advantages of GNSS Software Receiver 28](#_Toc387154432)

[3.3 Basics of GNSS Receiver Architecture 28](#_Toc387154433)

[3.3.1 RF Front End 30](#_Toc387154434)

[3.3.2 RF Front End Components 30](#_Toc387154435)

[3.4 Base Band Processing 35](#_Toc387154436)

[3.4.1 Acquisition 35](#_Toc387154437)

[3.4.2 Tracking 37](#_Toc387154438)

[3.5 Conclusion 38](#_Toc387154439)

[4. GLONASS C/A code Acquisition 39](#_Toc387154440)

[4.1 Introduction 39](#_Toc387154441)

[4.2 Acquisition Methods 39](#_Toc387154442)

[4.2.1 Serial Search in Time Domain 40](#_Toc387154443)

[4.2.2 Parallel Search in Frequency Domain 46](#_Toc387154444)

[4.3 Algorithm Implementation 52](#_Toc387154445)

[4.3.1 Coarse Acquisition 53](#_Toc387154446)

[4.3.2 Fine Frequency Acquisition 54](#_Toc387154447)

[4.4 Observations of Acquisition algorithm 54](#_Toc387154448)

[4.4.1 Results from Acquisition 56](#_Toc387154449)

[4.5 Conclusion 57](#_Toc387154450)

[5. Tracking 58](#_Toc387154451)

[5.1 Introduction 58](#_Toc387154452)

[5.2 Tracking 58](#_Toc387154453)

[5.2.1 Basic Demodulation of satellite signal 58](#_Toc387154454)

[5.3 Carrier Tracking Loop 60](#_Toc387154455)

[5.3.1 Basic Phase locked loop 60](#_Toc387154456)

[5.3.2 Costas loop 65](#_Toc387154457)

[5.4 Code Tracking Loop 68](#_Toc387154458)

[5.5 Combined Code Tracking and Carrier tracking loop 71](#_Toc387154459)

[5.6 Tracking Results 72](#_Toc387154460)

[5.7 Conclusion 73](#_Toc387154461)

[6. Time synchronous Implementation 74](#_Toc387154462)

[6.1 Introduction 74](#_Toc387154463)

[6.2 Time synchronous Implementation 74](#_Toc387154464)

[6.3 Time synchronous Pseudo code for Acquisition and Tracking in C 76](#_Toc387154465)

[6.3.1 Pseudo Code for Acquisition 76](#_Toc387154466)

[6.3.2 Pseudo Code for Tracking 83](#_Toc387154467)

[6.4 Description of C functions and structures used in Acquisition and Tracking 85](#_Toc387154468)

[6.4.1 Structures 85](#_Toc387154469)

[6.4.2 Functions 86](#_Toc387154470)

[6.5 Results 87](#_Toc387154471)

[7. Conclusion and Future Scope 90](#_Toc387154472)

[References 91](#_Toc387154473)

[Websites 92](#_Toc387154474)

[APPENDIX 93](#_Toc387154475)

# List of Figures

[Figure 2.1: Triangulation 16](#_Toc387237289)

[Figure 2.2: Structure of GLONASS systems 17](#_Toc387237290)

[Figure 2.4: GLONASS Transmitted Signal of nth satellite 19](#_Toc387237291)

[Figure 2.5: P – code generator 20](#_Toc387237292)

[Figure 2.6: GLONASS C/A code generator 21](#_Toc387237293)

[Figure 2.7: GLONASS signal spectrum 23](#_Toc387237294)

[Figure 2.8: Autocorrelation of 0.511MHz C/A code and its Delayed C/A code 26](#_Toc387237295)

[Figure 2.9: Auto correlation of 16 MHz sampled C/A code and its Delayed C/A code 27](#_Toc387237296)

[Figure 3.1: Basic software GNSS receiver 29](#_Toc387237297)

[Figure 3.2: Typical RF front end implementation 30](#_Toc387237298)

[Figure 3.3: Frequency domain depiction of the GLONASS signal and thermal noise power 31](#_Toc387237299)

[Figure 3.4: Frequency mixer 32](#_Toc387237300)

[Figure 3.5: Comparison of filters 33](#_Toc387237301)

[Figure 4.1: Serial Search Algorithm 40](#_Toc387237302)

[Figure 4.2: Cell – by – Cell Search method 41](#_Toc387237303)

[Figure 4.3: 511 chips of C/A code 43](#_Toc387237304)

[Figure 4.4: C/A code vs. Time 43](#_Toc387237305)

[Figure 4.5: Frequency domain representation of CA code 44](#_Toc387237306)

[Figure 4.7: 16000 samples of C/A code 45](#_Toc387237307)

[Figure 4.8: Spectrum of C/A code sampled at 16MHz 46](#_Toc387237308)

[Figure 4.9: Parallel frequency space search algorithm 47](#_Toc387237309)

[Figure 4.10: Parallel code phase search algorithm 49](#_Toc387237310)

[Figure 4.11: Acquisition of Satellite 3 at IF with Doppler frequency bin number 16 54](#_Toc387237311)

[Figure 4.12: C/A code begins at sample 6999 for GLONASS satellite 3 55](#_Toc387237312)

[Figure 4.13: Acquisition of satellite at IF 1.812MHz 56](#_Toc387237313)

[Figure 4.14: No satellite found for GLONASS satellite PRN 2 56](#_Toc387237314)

[Figure 5.1: Basic demodulation scheme to generate navigation data 59](#_Toc387237315)

[Figure 5.2: Simplified Tracking loop 60](#_Toc387237316)

[Figure 5.3(a): Time domain 61](#_Toc387237317)

[Figure 5.3(b): S-Domain 61](#_Toc387237318)

[Figure 5.4: linearized digital second-order PLL 63](#_Toc387237319)

[Figure 5.5: Second order phase lock loop filter 64](#_Toc387237320)

[Figure 5.6: Different loop responses depending on damping factor 65](#_Toc387237321)

[Figure 5.7: Costas loop block diagram 66](#_Toc387237322)

[Figure 5.8: Discrete time scatter plot 67](#_Toc387237323)

[Figure 5.9: Code tracking loop with six correlators 69](#_Toc387237324)

[Figure 5.10: Correlation of E, P, and L with incoming signal 71](#_Toc387237325)

[Figure 5.11: Simplified complete tracking channel of GLONASS Receiver 71](#_Toc387237326)

[Figure 5.12: Tracking results navigation data bits of Satellite PRN 3 72](#_Toc387237327)

# List of Tables

[Table 2.1: Exclusive operation and Multiplication 22](#_Toc387154476)

[Table 2.2: GLONASS carrier frequencies in L1 and L2 sub-bands 24](#_Toc387154477)

[Table 4.1: Comparison of Acquisition algorithms 50](#_Toc387154478)

[Table 4.2: Visible satellites parameters 57](#_Toc387154479)

[Table 5.1: Different types of code tracking loop discriminators 70](#_Toc387154480)

[Table 6.1: Comparison of Acquisition results in MATLAB and C 87](#_Toc387154480)

[Table 6.2: Comparison of Tracking results for 2msec in MATLAB and C 88](#_Toc387154480)

# List of Abbreviations

GPS Global Positioning System

GLONASS Global Navigation Satellite System

C/A Coarse Acquisition code

P-code Precision code

dBW decibel per watt

EIRPEquivalent Isotropically Radiated Power

LNA Low Noise Amplifier

DDS Direct Digital Synthesis

DDC Direct Digital Control

IF Intermediate Frequency

LO Local Oscillator

dB decibel

PLL Phase locked loop

ADC Analog to Digital Conversion

AGC Automatic gain control

FDMA Frequency Division Multiple Access

CDMA Code Division Multiple Access

ASIC Application Specific Integrated Circuit

DFT Discrete Fourier Transform

FFT Fast Fourier Transform

CLL Carrier Locked Loop

FLL Frequency Locked Loop

DLL Delay Locked Loop

# 1. Introduction

## 1.1 GNSS

Satellite navigation system with global coverage is called as Global Navigation Satellite System (GNSS). It is a [satellite](http://searchmobilecomputing.techtarget.com/definition/satellite) system that is used to find the geographic location of a user's receiver anywhere in the world. It allows small [electronic](https://en.wikipedia.org/wiki/Electronics) receivers to determine their location ([longitude](https://en.wikipedia.org/wiki/Longitude), [latitude](https://en.wikipedia.org/wiki/Latitude), and [altitude](https://en.wikipedia.org/wiki/Altitude)) with an accuracy up to less than one meter using [signals](https://en.wikipedia.org/wiki/Time_signal) transmitted along a [line-of-sight](https://en.wikipedia.org/wiki/Line-of-sight_propagation) path by transmitter from [satellites](https://en.wikipedia.org/wiki/Satellite). Satellite-based navigation systems use the principle of [triangulation](http://searchnetworking.techtarget.com/definition/triangulation) to locate the user through calculations using the information transmitted by a number of satellites where each satellite transmits coded signals.

Global Positioning System (GPS), developed and operated by the Department of Defense of United States of America, is the first GNSS system. GLONASS (GLObal NAvigation Satellite System) is another satellite based navigation system developed and operated by Department of Defense of Russia.

The original motivation for satellite navigation is from military applications. However, user countries all over the world which have been extensively using the GPS, or GLONASS for civilian applications have strongly felt the need to develop and launch their own regional/global navigation systems along with augmentation systems so that they may operate with or without collaboration from GPS or GLONASS. This has lead to the evolution of Global Navigation Satellite System that will have universal accessibility. GNSS refers not only to GPS, GLONASS, but also to Galileo (of Europe), COMPASS (of China), QZSS (of Japan) and IRNSS (of India) along with their respective augmentation systems.

Sometimes the GPS signals are not available to find out the position, due to blockage of signals by buildings, trees etc. Therefore, if a single receiver is developed which can be used to locate the position, using the signals of any GNSS system or two or more GNSS systems, availability and continuity of obtaining the receiver position increases because any four satellites of any system, which are visible can be used to locate the position.

## 1.2 GNSS Software Receivers

GNSS receivers have been traditionally implemented in hardware. A hardware GNSS receiver is conceived as a dedicated chip that have been designed and built with the only purpose of being a GNSS receiver. The presently available traditional GNSS receivers are hardware based and are available for both GPS and GLONASS.

GPS and GLONASS hardware based receivers are not flexible and cannot be easily upgraded to acquire the signals from other GNSS systems like Galileo and Compass, also the complexity of receiver increases because of the new signal structures and algorithms, introduced to improve the performance. Besides the new challenges of having the capability to receive and process the signals of many GNSS systems, some old challenges of GNSS like multipath still exist. Multipath is still one of the most dominant errors in positioning. However, research in the field of multipath navigation is active and is providing new receiver algorithms.

In general conventional GNSS receivers architecture can typically be partitioned into three distinct components. An analog section, responsible for the analog signal conditioning, the second component is a dedicated hardware subsystem (base band processor) to derive measurements from the received signal, and final component is a programmable processor, which is responsible for processing the accumulated measurements.

In a software GNSS receiver, all digital processing is performed by a general purpose [microprocessor](http://en.wikipedia.org/wiki/Microprocessor) or DSP or FPGA. In this approach, a small amount of inexpensive hardware is still needed, known as the [frontend](http://en.wikipedia.org/wiki/RF_front_end), which digitizes the signal from the satellites. The microprocessor can then work on this raw digital stream to implement the GNSS functionality. It also offers various advantages like,

* Software approach removes the nonlinear, temperature-dependent and age dependent components of the hardware receiver.
* A software receiver can provide more evolution and testing flexibilities. Some systems may collect complex data in both in-phase (I) and Qudrature (Q) channels while others use real data from one channel. The output data from these different platforms can be processed by the same software receiver. Both complex and real data can be generated by the software receiver with slight modifications .The software receiver can also adapt to digitized data with various sampling frequencies. The performance of different algorithms can be compared without any hardware development.
* A software receiver can provide researchers and developers with more evaluation, testing flexibility, and an effective simulation environment. New algorithms can be developed to solve problems such as jamming signals, without altering the hardware components.

The software receivers are more feasible particularly for the wireless environment compared to the hardware receivers. GNSS Software receivers allow a huge flexibility: many features of the receiver can be modified just through software. This provides the receiver with adaptive capabilities, depending on the user requirements and working conditions. In addition, the receiver can be easily upgraded via software. Under some assumptions, Software GNSS receivers can be more profitable for some applications. From these reasons, flexibility is seen as an asset for a software receiver.

## 1.3 GLONASS Software Receiver

At present GLONASS is the only alternative (to GPS) satellite based navigation system, operated by the Russian Space Forces. It is not only an alternative but also complementary to the presently existing United States GPS, or the upcoming Chinese [COMPASS](http://www.navipedia.net/index.php/COMPASS_General_Introduction) navigation system and the planned [GALILEO](http://www.navipedia.net/index.php/GALILEO_General_Introduction) positioning system of the European Union (EU). GLONASS has emerged as a pioneering way of determining one’s position, in recent years, with accuracy up to less than one meter. This provides reliable positioning, navigation, and timing services to users on a continuous worldwide basis freely available to all, although GLONASS is primarily intended for military users. It is more efficient than GPS for receiving high latitude signals.

However, there are up and downfalls in the history of GLONASS. It was conceived in the late 1960s. The government of the [Soviet Union](http://en.wikipedia.org/wiki/Soviet_Union) made a decision to develop the system in 1976. The first launch took place in 1982. Until its [dissolution](http://en.wikipedia.org/wiki/Dissolution_of_the_Soviet_Union) in 1991, the Soviet Union launched 43 GLONASS-related satellites. Work on the system was continued by the [Russian Federation](http://en.wikipedia.org/wiki/Russian_Federation) which brought its full operational capability in 1995. In the following years, the system fell into disrepair due to the economic crisis and diminished space funding. Starting from 2000, the government under President [Vladimir Putin](http://en.wikipedia.org/wiki/Vladimir_Putin) made the restoration of GLONASS a top priority, its funding was doubled and after a several years. In 2003, a new generation of satellite design GLONASS – M was introduced. By the end of 2005, 14 satellites were active in orbit. As of 2010, launches of GLONASS – M satellites continue until 2012. By early 2011, GLONASS had 22 operational satellites, two less than the required constellation of 24 to provide global coverage. By the fall of the same year, 23 GLONASS satellites were fully operational. The latest and significantly improved satellite design [GLONASS-K](http://en.wikipedia.org/wiki/GLONASS-K) was introduced in 2011. In April 2013, the Russian Ministry of Defense reported 29 satellites in the GLONASS network including 23 operational spacecrafts and two under a temporary out of service for technical maintenance. Three additional satellites served as backups and a single [GLONASS – K](http://www.russianspaceweb.com/uragan_k.html) spacecraft was undergoing flight testing [11].

Development of individual software receivers for each GNSS system GPS, GLONASS, Galileo, is the first step for having common software receiver for all systems. At present many open source GPS software receivers are available [GNSS-SDR, OpenGPS, SoftGPS etc]. Of course Galileo, Compass, IRNSS systems are under development only. Scilab based GLONASS software receiver available at [GLONASS SDR] [20].

## 1.4 Motivation

The availability of real-time satellite signals used in software receivers has opened up a wide range of research opportunities in GNSS world. Many commercial devices use GPS navigation system and still a lot of research is going on this, but there is an alternative system, available i.e. GLONASS but still a lot of development is to be done at receiver part to get into real time. Therefore a development of GLONASS Software Receiver is required. The presently available scilab based GLONASS software receiver cannot be used directly for real time application [20]. This software receiver first takes the complete available IF data and processes for acquisition, tracking and demodulation. Later, it will find out the position of receiver for every milli second. Therefore it can be neither used for real time nor for time synchronous processing. It will be useful for offline analysis of IF data for computing the receiver position. However it can be converted in to Time synchronous with few modifications and later as a real time software receiver.

## 1.5 Objective of the Project

The main objective of this thesis is to implement acquisition and tracking algorithms of GLONASS software receiver in C. In this design, the focus is on the time synchronous tracking and demodulation of L1 carrier, C/A code of GLONASS signals. The MATLAB software receiver code was taken as reference and converted to time synchronous software receiver in C. The converted code is running successfully in time synchronous mode and results are exactly matching with MATLAB based software receiver. The framework and algorithms implemented for developing the Time synchronous software receiver in C are explained in this report in detail.

## 1.6 Organization of the Report

The next chapter i.e. second chapter discusses about the GLONASS Architecture and its signal structure. GLONASS architecture describes about system design and signal structure part describes specifications of glonass signals and also describes the correlation properties of the C/A code.

Third chapter discusses about general view of GNSS Software receiver, RF front end and base band processing of software receiver and also about significance of acquisition, factors effecting acquisition, performance of tracking and navigation.

Fourth chapter describes about several related acquisition algorithms and methods, which are used in conventional hardware receivers and software receivers.

Chapter five describes about tracking. In tracking module, two tracking loops are utilized for demodulation of navigation data bits which have been explained and implemented in C.

Chapter six discusses about time synchronous GLONASS software receiver in C and also describes the procedure for Implementation of this time synchronous code.

The seventh chapter describes about conclusion and future work.

# 2. GLONASS Architecture and Signal Structure

## 2.1 Introduction

The basic principle of operation, Architecture, and signal structure of GLONASS are essential to develop a GLONASS receiver, which are explained in this chapter.

## 2.2 Principle of Operation

The basic working principle of [GLONASS navigation system](http://www.accessbuy.com/GPS-Tracking-Devices.html) is to measure the distance between the known satellite and GLONASS receiver at user side, then it analyses and confirms the specific geographic position of the GLONASS receiver. To calculate the position of the receiver, timing information is required. The time information is placed in the signal and broadcasted by the satellite. The signal contains data that a receiver uses to compute the locations of the satellites and to make other adjustments needed for accurate positioning.

Suppose the GLONASS signal travels to the ground at the speed of light, still it takes a measurable amount of time to reach the receiver. The receiver then calculates the distance to the satellite by measuring the difference between the time when the signal is received and the time when it was sent, and then multiplying with the speed of light. The receiver must account for propagation delays or decrease in the signal’s speed caused by the atmosphere. The received signal is needed to extract the navigation message and timing information to determine the location using either hyperbola or triangulation principles. This is the fundamental principle of GLONASS.

Propagation time = Time signal reached receiver – Time signal left satellite

Distance = speed of light × Propagation time

Triangulation is the process, which is based on the location of satellites in space as reference points. To determine the user position, three satellites and three distances are required that is shown in Figure2.1. Imagine an unknown person standing somewhere on Earth with three satellites above him in space. If he knows how much is the distance from satellite A, then he will be located somewhere on the sphere. If the same is the case for satellites B and C, he will be located where the three spheres intersect. The intersection of these three distances should indicate the user’s position.

##### C:\Users\R.PAVAN KUMAR REDDY\Desktop\3sat.jpg

##### Figure 2.1: Triangulation

However three satellites are needed to determine latitude and longitude, while a fourth satellite is necessary to determine altitude without any clock error. However the satellite and receiver are controlled by separate clocks. The satellites are set as accurately as possible with an atomic clock, and are assumed to be synchronized with one another. In order to measure distance accurately the time measured is to be precise as an error of one microsecond in the synchronization of the two clocks creates an error of 300 meters. By taking a measurement from a fourth satellite, the receiver avoids the need for an atomic clock and provides accurate position. Thus, the receiver uses four satellites to compute latitude, longitude, altitude, and time.

## 2.3 Architecture of GLONASS

The architecture of GLONASS system consists of three segments:

(i) The Space segment

(ii) The Control Segment

(iii) The User segment

The structure of GLONASS system is shown in figure 2.2. All these segments operate together to provide accurate three-dimensional positioning, timing and velocity data to users worldwide.

### 

GLONASS Control channels

GLONASS Broadcasted signals

##### Figure 2.2: Structure of GLONASS systems

**(i) Space Segment**

The Space Segment of the GLONASS system is located in space and it consists constellation of all satellites. These GLONASS satellites are distributed over three orbital planes with an inclination of 64.8° (angle between equator and satellite orbit is called inclination angle) and each orbit has eight satellites. The longitude of ascending node differs by 120° from plane to plane. These GLONASS satellites continuously broadcast ephemeris data which provides satellite ranging and timing information, and also to store and retransmit the navigation message sent by the control segment. These transmissions are controlled by highly stable atomic clocks on board the satellites [10].

**GLONASS Satellite Constellation**

The GLONASS system uses a satellite constellation consisting of 24 satellites at an altitude of 19100km above the earth’s surface. These satellites are arranged in a set of 8 satellites in 3 orbital planes as shown in figure 2.3.

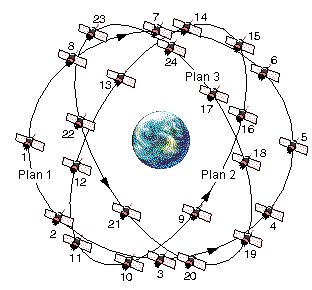
[](http://www.navipedia.net/index.php/GLONASS_Space_Segment)

Figure 2.3: GLONASS Satellite Constellation

Each satellite is identified by its slot number, which defines the orbital plane and its location within the plane. These satellites kept at such heights provide at least four visible satellites to user at any location and at any time on the earth. A minimum of four satellites in view allows a GLONASS receiver to compute its position in three dimensions [14].

**(ii) Control Segment**

The [GLONASS](http://www.navipedia.net/index.php/GLONASS_General_Introduction) Control Segment (also referred to as Ground Segment or Operational Control System) is responsible for the proper operation of the system. It includes data upload stations, System Control Center and the network of the Telemetry, Tracking and Control (TT&C) Stations that are located throughout the territory of Russia to ensure maximum satellite coverage and ground antennas.

The operations of control segment include maintaining satellite orbital position and monitoring the health of the satellite constellation. The health includes parameters like the power, fuel levels etc. It determines the ephemerides and satellite clock offsets with respect to GLONASS time and UTC (Universal Time Coordinated) and uploads the navigation data to the satellites twice a day. The ground stations make pseudo range measurements by passively tracking the satellites. This data is used by the master control station to update the navigation message with ephemeris data, corrections and almanac data. This updated information is called TT&C (Telemetry, Tracking and Command) data. This information for each satellite is uploaded by a ground uplink antenna when that particular satellite is in view of the antenna [14].

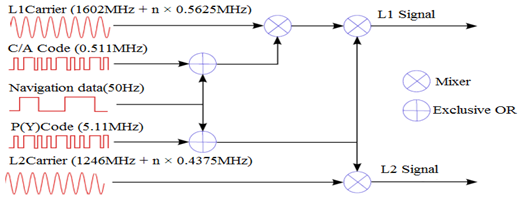
**(iii) User Segment (Receiving Segment)**

The [GLONASS User Segment](http://www.navipedia.net/index.php/GLONASS_User_Segment) consists of L-band radio receiver/processors and antennas which receive [GLONASS](http://www.navipedia.net/index.php/GLONASS_General_Introduction) navigation signals and utilizes these signals to find out the user’s location on surface of the earth using triangulation principles.

The GLONASS receivers employ the hemispherical coverage which has a Right Hand Circular Polarization (RHCP). The polarization ensures the differentiation between the multipath and direct path signals. The GLONASS receiver determines pseudoranges and solves the navigation equations in order to obtain their coordinates and provide a very accurate time. It does this by calculating the code and carrier phases and demodulating the navigation message data.

## 2.4 GLONASS Signal Structure

The GLONASS signals are transmitted on two different frequencies, L1 and L2 carrier sub-bands. These satellite generated carrier frequency of L1 and L2 sub bands are modulated by the modulo-2 addition of pseudo random noise code, navigation data using Binary Phase Shift Keying (BPSK) modulation technique. GLONASS satellite broadcasted signals include ranging signals, used to measure the distance to the satellite, and navigation messages. However, in GLONASS each satellite transmits the same ranging code signal on different carrier frequencies using Frequency Division Multiple Access (FDMA) technique.



##### Figure 2.4: GLONASS Transmitted Signal of nth satellite

The mathematical equation of GLONASS satellites transmitted civilian signal (C/A code) on L1 band has the following structure

........................Eq 2.1

Where: Sn = GLONASS Broadcast on nth frequency channel

n = Indicates the frequency channel number

P = Signal Power

PRN = PRN code (511kcps)

Dn = Navigation Data for the nth frequency channel

Fc = 1602.0MHz (GLONASS Nominal Frequency (zero channel))

Ø = Phase offset

Any GLONASS satellite broadcasted signal consists of three basic components

* Ranging code
* Carrier
* Navigation data

**Ranging code**: Also called as Pseudo random Noise code. It is a random binary sequence having special properties. There are basically two types of standard ranging signals generated by GLONASS satellites. They are:

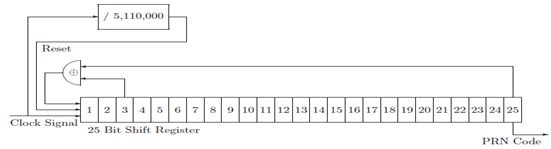
* Precision Code (P-code) (or) High accuracy signal.
* Coarse (or Clear) Acquisition code (C/A-code) (or) Standard accuracy signal.

### 2.4.1 Precision Code (P - code)

The Precision code with clock rate of 5.11 MHz is encrypted by special code. This P code at 5.11MHz is bi-phase (BPSK) modulated on L2 carrier frequency, therefore the main lobe of the spectrum is 10.22 MHz wide from null to null. The chip length is about 0.195µs (1/5.11MHz).

The Precision code is not directly transmitted by satellite, it is encrypted by Y-code so it is often referred as P(Y) code. The P(Y) code is not available to civilian users and is primarily used by military. The P(Y) code has similar properties as that of P-code. It can also be called as High precision service (HPS) code (or) High accuracy signal [2].

A functional block diagram of GLONASS P- code generator is shown in below.



##### Figure 2.5: P – code generator

Satellite generated P – code comprises of 25 bit shift register with a tap feedback that would produce a 33554431 – bit maximum length sequence, except for the fact that, it is short cycled to 5, 11,000 bits. It is clocked at a rate of 5.11MHz so that it repeats once per second. The initial state is defined such that each bit contains the value ‘1’. All satellites generates a same P-code and this code is modulated on L2 sub bands.

The P – code can be described by the polynomial mentioned below

1 + x3 + x25......................................................Eq 2.2

All satellites broadcast civilian signals on both L1 and L2 carrier signals. The L1 band carrier signal is modulated by the C/A code; the L2 carrier signal is modulated by P-code and C/A code. Now our focus is on the L1band modulated C/A code of GLONASS satellite system described in next section.

### 2.4.2 Coarse Acquisition Code (C/A code)

In GNSS the C/A (Coarse or Clear acquisition) codes are primarily used for broadcasting civilian signals on L1 band radio frequency with a clock rate 0.511 MHz, therefore the main lobe of the spectrum is 1.022MHz and is shown in figure below. It can be freely available to civilian users worldwide for positioning receiver on the earth [2].

The GLONASS C/A code belongs to the family of pseudorandom noise (PRN) codes known as the M-codes.C/A code can also called as ranging code and it is used for standard positioning service, so it can also be called as standard accuracy ranging code. All GLONASS satellites use the same C/A-code. The pseudorandom noise (PRN) codes transmitted by the GLONASS satellites are deterministicsequences having a special noise like properties. Each C/A code is generated using a tapped 9 bit linear feedback shift register (LFSR). It generates a maximal length sequence of length L = 2n – 1. The C/A code generator of GLONASS satellite system is shown in Figure2.6.

1 2 3 4 5 6 7 8 9

Clock signal 0.511MHz

PRN code

##### Figure 2.6: GLONASS C/A code generator

The C/A – code can be described by the polynomial mentioned below

1 + x5 + x9.................................................................. Eq 2.3

The nominal values of L1 sub band carrier frequencies are defined by following expressions

………………………… Eq 2.4

Where ‘n’ (-7 to 6) is the frequency channel of signals transmitted by GLONASS satellites in the L1 and L2 sub band.

Fc =1602 MHz for L1 sub band

FL1 = GLONASS satellite broadcasted carrier signal on L1 sub band.

n = frequency slot number or PRN number.

The C/A code as employed by the GLONASS satellite system is a Pseudo Random Noise sequence of binary digits or chips derived from the seventh bit of nine bit shift register. Let us assume a 9 bit shift register containing all ‘1’s (Let us Assume ‘0’ as ‘1’ and ‘1’ as ‘-1’). The C/A code signal is generated from the 7th bit of nine bit shift register. The feedback values are taken from 5th and 9th bits of shift register; these two bits are XOR operated (when it is ‘0’ and ‘1’), but here multiplied (as it is ‘1’ and ‘-1’). The produced output is fed back to 1st bit of shift register. Thus a maximum length sequence generator can be made from shift register with proper feedback. The resultant signal and data signal perform modulo-2 addition technique and spread the data signal. The spreaded signal is modulated on particular frequency channel of L1 sub band using BPSK modulation. The operating rule of the modulo-2 adder and Multiplication is shown in below.

|  |  |  |
| --- | --- | --- |
| Input1 | Input2 | Output |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

###### Table 2.1: Exclusive operation and Multiplication

|  |  |  |
| --- | --- | --- |
| Input1 | Input2 | Output |
| -1 | -1 | 1 |
| -1 | 1 | -1 |
| 1 | -1 | -1 |
| 1 | 1 | 1 |

The C/A code signal bandwidth for GLONASS is approximately half in comparison to GPS signal. Because in GPS each C/A code signal has 2.044MHz bandwidth from null to null but in GLONASS, the ranging code signal has 1.022MHz bandwidth from null to null.

**Carrier**

Each GLONASS satellite transmits navigation signals in two sub-bands of L-band (L1 1.6 GHz and L2 1.2 GHz) frequencies. In some situation two GLONASS satellites may transmit same carrier frequency if they are located in antipodal slots of a single orbital plane.

GLONASS system uses FDMA technique it means each satellite generates its own carrier frequency with a small amount of step size in the L1 and L2 band of frequencies. However eachsatellite is assigned oneof the 24 frequencies with a step size of 0.5625MHz using 15 channel frequency divisions multiple access (FDMA) spreadedover the 14 MHzband at intervals of562.5 kHz, so bandwidth of GLONASS satellite system in L1band frequency is approximately 13MHz. The carrier signal with frequency FL1 or FL2 is modulated by the Modulo-2 addition of Pseudorandom (Ranging)code and digital data of navigation message.

### 2.4.3 Frequency plan

The frequencies of L-band is divided in to two bands that are: The L1 sub-band ranges from “1598.0 MHz to 1605.5MHz” in steps of “0.5625MHz” while L2 sub- band ranges from “1242.9375MHz to 1248.625MHz” in steps of “0.4375MHz”. The frequency spectrum of GLONASS system is shown in below figure 2.7. The nominal values of L1 and L2 carrier frequency are defined by the following expressions:

………..…………………………Eq 2.5

………..…………………………Eq 2.6

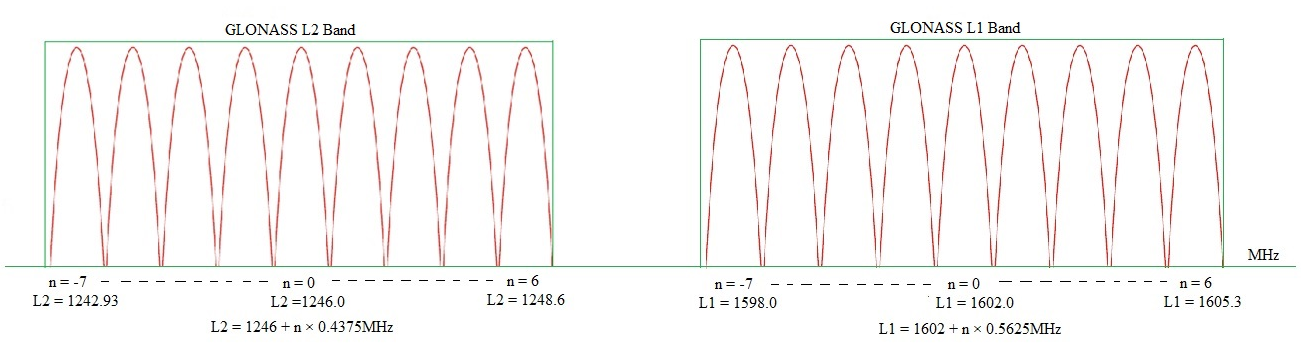


Figure 2.7: GLONASS signal spectrum

Where ‘n’ is the frequency channel of signals transmitted by GLONASS satellites in the L1 and L2 sub band.

Fc1 = 1602MHz for L1 sub band and Fc2 = 1246MHz for L2 sub band.

FL1 = nominal frequency of zero frequency channel 1602 MHz of L1 sub band

FL2 = nominal frequency of zero frequency channel 1246 MHz of L2 sub band

The frequency plan of GLONASS satellites generated carrier frequencies on L1 and L2 sub bands are depicted in table below [8].

###### Table 2.2: GLONASS carrier frequencies in L1 and L2 sub-bands

|  |  |  |  |
| --- | --- | --- | --- |
| No. of  Channel | Nominal value of frequency in L1 sub-band, MHz | No. of  channel | Nominal value of frequency in L2 sub-band, MHz |
| 06 | 1605.375 | 06 | 1248.625 |
| 05 | 1604.8125 | 05 | 1248.1875 |
| 04 | 1604.25 | 04 | 1247.75 |
| 03 | 1603.6875 | 03 | 1247.3125 |
| 02 | 1603.125 | 02 | 1246.875 |
| 01 | 1602.5625 | 01 | 1246.4375 |
| 0 | 1602.0 | 0 | 1246.0 |
| -01 | 1601.4375 | -01 | 1245.5625 |
| -02 | 1600.8750 | -02 | 1245.1250 |
| -03 | 1600.3125 | -03 | 1244.6875 |
| -04 | 1599.7500 | -04 | 1244.2500 |
| -05 | 1599.1875 | -05 | 1243.8125 |
| -06 | 1598.6250 | -06 | 1243.3750 |
| -07 | 1598.0625 | -07 |  |

The channel number ‘n’ of any particular GLONASS satellite is provided in the almanac of navigation data. It is a unique integer for each satellite and varies from -7 to 6. The plan is to have satellites on opposite sides of the Earth (antipodal) so that they can share the same frequency number.

**Navigation data**

The GLONASS navigation message is generated as continuously repeating super-frames. A super-frame consists of frames and a frame consists of strings. The navigation data contains information regarding satellite orbits, clock information. This information is uploaded to all satellites from the ground stations in the GLONASS Control Segment. The navigation data have a bit rate of 50 bps.

## 2.5 Correlation Properties of C/A Code

The most important characteristics of the C/A codes are their correlation properties. The auto correlation of sequence is correlation of a sequence with itself. The auto correlation of a sequence x (n) is defined by

……………..……………………... Eq 2.7

If the time shift m=0 the we have

……………………………………………....Eq 2.8

Cross correlation can be defined as correlation of two different signals, but in GLONASS cross correlation is not applicable because here all satellites uses same C/A code or PRN code so only auto correlation is applicable. But in GPS the cross correlation properties are required because it uses different PRN codes for all satellites so here the cross correlation gives the low cross correlation peak as two different signals cannot match so that the resulting power is very low. If the codes are orthogonal the cross correlation result is zero but the GPS C/A codes are GOLD codes, these are not orthogonal code but near orthogonal codes.

The auto correlation function of C/A code is

……………………………………….Eq 2.9

If the time shift t=0

…………………………………..……….Eq 3.0

Where: x (n) is the satellite generated C/A code.

x (n + t) is locally generated C/A code with time shift.

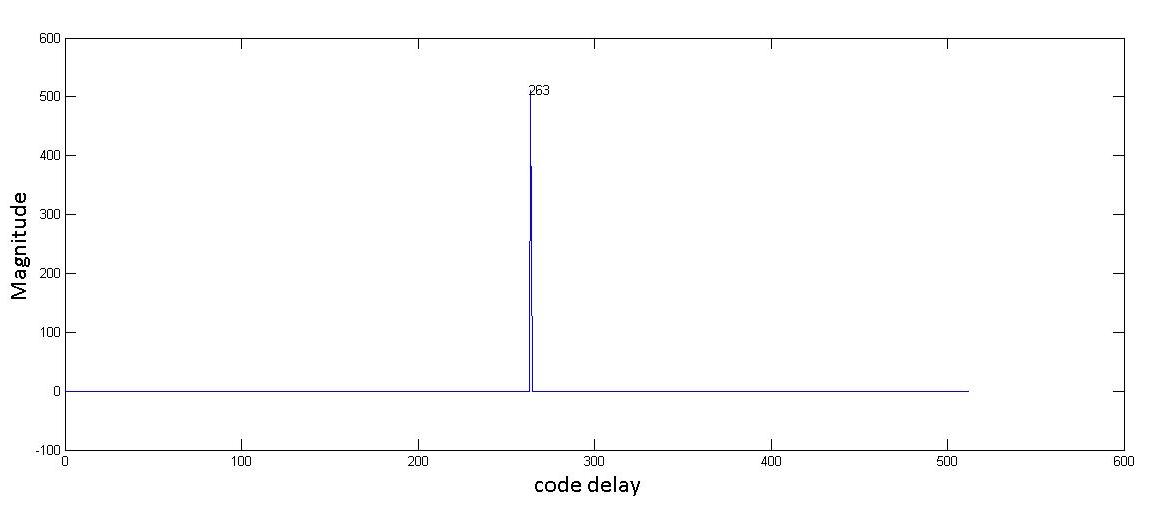
The correlation peak repeats every code period. The high correlation peak property of the Autocorrelation function is used to synchronize the receiver locally generated code with the code of the received signal.

High autocorrelation peak and low cross-correlation peaks can provide a wide dynamic range for signal acquisition. In order to detect a weak signal in the presence of strong signals, the autocorrelation peak of the weak signal must be stronger than the cross-correlation peaks from the strong signals.

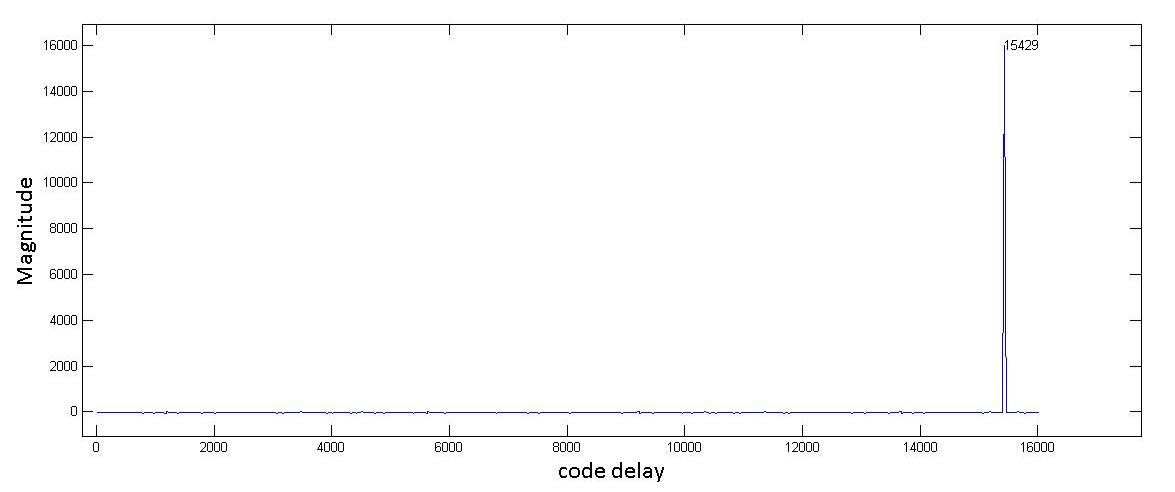
To remove the ranging code from the incoming signal correlation is required. Locally we generate the same ranging code but delayed; when both codes match then correlation will give the peak value as the delay between incoming signal and locally generated ranging code is zero.

GLONASS satellite system uses the M-codes, these codes are very easy to generate and have excellent auto correlation properties but have poor cross correlation properties.

Thus we can generate different signals with different delay values of same C/A code signal for the periodic autocorrelation results. The results are changing for different delayed signals of same C/A code signal, and then the auto correlation of those different delayed signals will produce different peak values.



##### Figure 2.8: Autocorrelation of 0.511MHz C/A code and its Delayed C/A code



##### Figure 2.9: Auto correlation of 16 MHz sampled C/A code and its Delayed C/A code

The above Figure 2.9 shows the correlation between 16MHz sampled C/A code and locally generated delayed version of C/A code. Therefore it gives maximum value or peak value when both codes are well aligned. However, the graph explains that the orginal signal is well algined after 15429 bits long with the delayed version of same signal. Thus it will produce some amount of bits lag of original signal due to the time delay between two signals. Thus the auto correlation can be performed.

## 2.6 Conclusion

This chapter explained about the introduction of GLONASS architecture and principle operation of all segments and also discussed about GLONASS satellite signal structure, frequency allocation of channels in L band frequencies, correlation properties of C/A code. These are very useful to understand base band processing of GLONASS software receiver in next chapter.

# 3. GNSS Software Receiver

## 3.1 General overview of GNSS Receiver

Global Navigation Satellite System is a satellite-based navigation system that has been used widely in both civilian and military for positioning, navigation, timing and other position related applications. The hardware-based GNSS receivers provide less user flexibility, so it is necessary to have Software-based GNSS receivers for easy and quick implementation, simulation and analysis of algorithms. Software-based GNSS receiver processes the output of RF front end of GLONASS or GPS, which is at radio frequency or intermediate frequency that depends on the requirement of the receiver. In these software receivers, different modules are available which will be introduced in this chapter.

## 3.2 Advantages of GNSS Software Receiver

A GNSS software receiver has more flexibility due to its hardware independence. The receiver is mainly implemented in software except for the front-end part, which offers various advantages. First, a software receiver removes the nonlinear, temperature-dependent components of conventional hardware receivers. Second, a software receiver can provide more evaluation and testing flexibilities. Some systems may collect complex data in both the in-phase (*I*) and Qudrature (*Q*) channels while others use real data from one channel.

The software receiver can also adapt to data digitized with various sampling frequencies. The performance of different algorithms can be compared without any hardware development. Third, it provides an effective simulation environment. New algorithms can be developed for the software receiver to solve problems, such as jamming signals, without altering the hardware components.

In order to properly calculate the user position, at least 30 seconds of GLONASS data is required. However, this is another advantage in a software receiver, where only 30seconds of data can be used to calculate an initial user position.

## 3.3 Basics of GNSS Receiver Architecture

A basic software-based GNSS receiver composes of two main parts, as shown in Figure 3.1. The first part covers the hardware section which can be divided in two functional blocks: RF section and IF section. The RF section consists of analog hardware modules, responsible for RF to IF conversion, while the IF section composes of digital hardware modules. The second part is the baseband software processing, which performs acquisition, tracking and Navigation decoding of the received signals, as well as the necessary algorithms used for the computations of the receiver PVT.

LNA

Reference Oscillator

Frequency Synthesizer

Down Converter

Analog IF

Filter

AGC

A/D Converter

Digital IF

Antenna

Hardware Section

Navigation Decoding

Tracking

Acquisition

Software Section

PVT solution

##### Figure 3.1: Basic software GNSS receiver

The operation of receiver section consists of four blocks: antenna, RF front end, local oscillator, base band signal processing block. The antenna is the first element of receiver architecture which will make the first signal conditioning. The transmitted signal from satellites is Right Hand Circularly Polarized (RHCP), so the antenna must be set to receive RHCP signals. The process starts with the signals transmitted from the satellites, propagating through space with the velocity of light, and incident on a GNSS’s antenna. The antenna gain pattern is an important consideration that indicates how well an antenna performs at different centre frequencies, polarizations and elevation angles. Finally, all the measurements on radio waves i.e., time-delay measurements, phase measurements or doppler-shift measurements can be performed on the receiver side.

The power level of the received RF signal from GLONASS satellite at the output of a 3dBi (The db"i" is a measurement of an "isotropic" antenna, in an ideal situation, an antenna that radiates equally in all directions) linearly polarized antenna is not less than -161 dBW for L1 sub-band provided that the satellite is observed at an angle of 5° or more. The radio signals collected by an omnidirectional receiving antenna are weak. As GNSS signal’s have low power at reception, after the antenna set filtering and low-noise amplification stages to be performed.

### 3.3.1 RF Front End

A generic RF front end consists of amplifiers, filters, and mixers. A typical RF front end is shown in figure 3.2. After receiving signal from antenna it is then forwarded and processed in the RF front-end. In the RF front-end of GNSS receiver first component is preamplifier after antenna; it is often used very close to antenna. The antenna may be capable of receiving multiple frequency bands. After amplification of the signal coming from the preamplifier, it is sent to down conversion part to convert input signal from RF to a low IF.

LNA

Mixer

Filter

L.O

Antenna

filter

##### Figure 3.2: Typical RF front end implementation

This down-conversion is accomplished by mixing the input RF signal with the local oscillator (LO) generated signal, which must be carefully chosen to avoid harmonics and image frequencies near IF, using one or two mixers followed by an appropriate band pass filtering. The signal having Doppler and the PRN codes are preserved after the mixing process, only the carrier frequency is lowered.

The resulting analog IF signal is then converted to a digital IF signal through the analog-to-digital converter (ADC). Here ADC sampling frequency, Fs should be carefully chosen. In the conversion process of analog to digital, received analog signal consist slow variation in amplitude so that AGC is required to continue adjustments in the receiver’s gain in order to maintain a relative constant output signal. The sampling frequency is to be selected to convert the IF frequency directly to base band. The IF digital signal is then transmitted to a post processing unit or base band processing block, which will handle all the baseband processing and necessary computations to achieve the receiver’s PVT.

### 3.3.2 RF Front End Components

The process begins with the GNSS signal, propagating through space, which is incident on a user’s GNSS antenna. This, in turn induces a voltage within the element. That voltage is extremely weak, corresponding to a guaranteed signal power of −161 dBW in the case of the Global Navigation Satellite System (GLONASS) [8] and has a carrier frequency of L1 (1598MHz – 1605.5MHz) subband.

Considering a bandwidth of 1MHz (the approximate null-to-null bandwidth the GLONASS C/A code signal), the received GLONASS signal power is actually below that of the thermal noise floor, as defined by Equation 3.1 with a simplified illustration in Figure 3.4.

Let the Boltzmann’s constant be denoted by *k* = 1*.*38×10 -23 J/°K, the absolute temperature by *t* in ◦ K, and the equivalent noise bandwidth by *B* in Hz, then

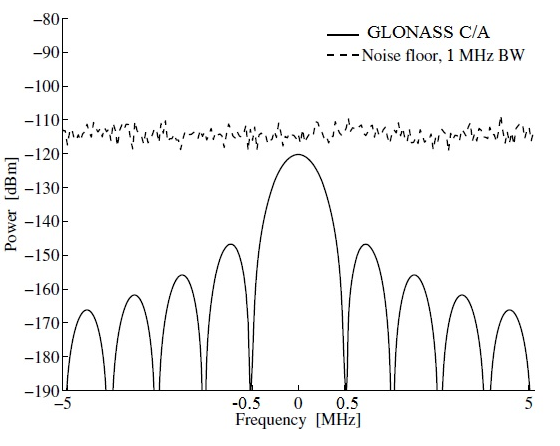
*P* Thermal Noise = *k t B*........................................................................... Eq3.1

For the GLONASS C/A code signal *P* Thermal Noise can be approximated by

1*.*38×10 -23 ×290 × 1 × 106 = 4.002 × 10-15 or

More conveniently expressed in dB as

10 × log10 (4.002 ×10-15) = −161 dBW= −131dBm



##### Figure 3.3: Frequency domain depiction of the GLONASS signal and thermal noise power

**3.3.2.1 Low Noise Amplifier**

Low-noise amplifier (LNA) is an [electronic amplifier](http://en.wikipedia.org/wiki/Electronic_amplifier) used to amplify possibly very weak signals (for example, captured by any active or passive [antenna](http://en.wikipedia.org/wiki/Antenna_(electronics))). It is also called as Preamplifier. It is usually located very close to the detection device to reduce losses in the [feed line](http://en.wikipedia.org/wiki/Feedline). The antenna may be capable of receiving multiple frequency bands. Thus, there may be one pre-amplifier per frequency band of interest, or a single pre-amplifier may cover multiple frequency bands. The primary purpose of the preamplifier is to amplify the signal at the output of the antenna for further processing. However, this amplifier has a very low noise figure so it is usually referred as a Low Noise Amplifier (LNA) [4].

**3.3.2.2 Mixer**

It is used in any communication system to change the carrier frequency of a modulated signal. Mixer is a non-linear device and it does the job of frequency translation of signals from one frequency to another frequency in Transmitters and Receivers.

Carrier signal increment - Up conversion and Carrier signal decrement - Down conversion

These both can be done by Mixer so that RF signal can be shifted to FC + FL and FC - FL.

Incoming signal

IF signal

Filter

Local Signal

Mixer

##### Figure 3.4: Frequency mixer

A simplified block diagram of a frequency mixer is shown in above Figure 3.4. At the mixer output, a sinusoidal signal with an intermediate frequency (IF) appears. In fact, this Intermediate frequency is the difference between incoming signal frequency and the locally generated signal frequency (IF = FRF - FLO).

Band pass filter is used to select up convert or down convert signal. Mixer is represented by the symbol shown in Figure 3.4. It consists of two input and one output ports, namely RF, LO and IF ports. LO port is always input port. In up-conversion application IF port is input port and RF port is output port. Whereas in down-conversion application it is vice-versa, i.e., RF port is input port and IF port is output port.

Consider two input signals Vi1 = A1 cos (ω1t) and Vi2 = A2 cos (ω2t) to the mixer then the output is given as,

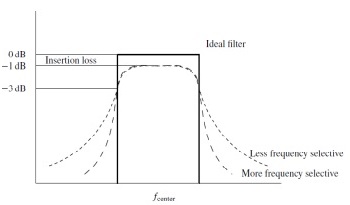
HF Component

IF Component

Out of the addition and subtraction of frequency components, in Receiver application (where the frequency needs to be down-converted) the difference of the components is utilized, whereas in Transmit application (where the frequency needs to be up-converted) the addition of the components is utilized. The un-utilized component, in either of the applications, which is to be rejected is called image component and can be filtered by using the band pass filter. IF component is the desired component of mixer output. High Frequency (HF) component is undesired and filtered out.

**3.3.2.3 Filter**

A filter is a frequency selective device that allows only certain frequencies to pass and attenuates others. Here filter is a band pass filter, as opposed to a low pass or high pass filter, and its purpose is to provide additional frequency selectivity. Filters can be characterized by their *insertion loss*, or the attenuation of the desired frequency components. For ideal conditions insertion loss nearly equals to zero, but in practical conditions, it should be as low as possible.



##### Figure 3.5: Comparison of filters

The second filter parameter is the *bandwidth*. Again, since no filter is ideal, typically a 3 dB bandwidth is specified. This indicates the frequency(s) attenuation will be 3 dB (or 50% of the signal power). However, these two parameters cannot completely describe most of the filters but only provide some insight into their performance shown in Figure 3.5. A goal in filter design is to provide sharp transitions between the desired (pass band) and undesired (cut-off) frequencies while maintaining a minimal insertion loss [4].

**3.3.2.4 Amplifier**

Amplification is the process that increases the signal magnitude. Unlike most filters, an amplifier is an active component and requires power to accomplish its function. The ideal amplifier would only increase the amplitude of the signal. However, any practical amplifier will not only increase the amplitude but also add noise to the resulting signal. The fundamental parameters used to describe an amplifier are:

1. *Gain*, usually expressed in dB, and often assumed constant over a Specified *frequency range*.

2. *Noise figure*, again usually expressed in dB, and indicative of the amount of noise that will be added to the signal being amplified [4].

**3.3.2.5 Intermediate Frequency**

In [electronics and communication engineering](http://en.wikipedia.org/wiki/Electronic_engineering), an intermediate frequency (IF) is a [frequency](http://en.wikipedia.org/wiki/Frequency) to which a [carrier frequency](http://en.wikipedia.org/wiki/Carrier_wave) is shifted which is an intermediate step in [transmission](http://en.wikipedia.org/wiki/Transmission_(telecommunications)) or reception.  The intermediate frequency is created by mixing the carrier signal with a [local oscillator](http://en.wikipedia.org/wiki/Local_oscillator) generated signal in a process called [heterodyning](http://en.wikipedia.org/wiki/Heterodyning), resulting in a signal at a different or [beat frequency](http://en.wikipedia.org/wiki/Beat_frequency).

**3.3.2.6 Local Oscillator**

The local oscillator for GNSS front-end design is typically a combination of components. Mostly crystal oscillators are used, either standalone or temperature compensated for greater stability. These are not capable of generating the desired local oscillator frequency for the L1 GNSS signal. So that, a phase lock loop (PLL) is combined with the crystal to achieve the desired higher frequency of the local oscillator. In addition, it is a common practice that the local oscillator being divided down to serve as the sampling clock.

**3.3.2.7 Analog to Digital Converter (ADC)**

The final component in the RF front-end hardware part is the analog-to-digital converter. It does the conversion of the analog signal to digital samples. The IF signals is sampled and digitized through the ADC. The resulting samples are processed by signal processor. The key parameters of ADC are the number of bits, the maximum sampling frequency, the analog input bandwidth, and the analog input range.

In the conversion process of continuous signal to digitized signal, ADC having a feed back of automatic gain control (AGC) loop controls the amplitude of the desired incoming signal. The function of the AGC is to monitor samples of input signal level and gives fixed amplitude levels of output to the ADC to ensure that it meets the dynamic range requirements of the system.

In GLONASS case the input signal to the ADC is at an IF of approximately -3.5MHz, and the sampling frequency of ADC is 16MHz. Therefore the sampled output of ADC is processed by base band signal processing using software acquisition, Tracking and Navigation decoding.

## 3.4 Base Band Processing

The base band processing mainly consists of 3 blocks:

* Acquisition
* Tracking
* Navigation datadecoding

### 3.4.1 Acquisition

Generally GNSS receiver must know which satellites are currently visible. There are two common ways to identify the initially visible satellites. One is referred to as warm startand the other is cold start.

**Warm start:** In a warm start, the receiver combines information in the stored almanac data of the last position computed by the receiver. The almanac data is used to compute coarse positions of all satellites at the actual time. These positions are then combined with the receiver position in an algorithm computing which satellites should be visible. The warm start has at least two downsides. If the receiver has moved far away (e.g., to another continent), the receiver position cannot be trusted and the found satellites do not match the actual visible satellites since it was turned off. Another case is that the almanac data can be outdated, so they cannot provide good satellite positions. In both cases, the receiver has to make a cold start.

**Cold start:** It takes longest time when the receiver does not depend on any stored information, so it starts from searching of satellites. The method of searching is referred to as acquisition, for the purpose of identifying the satellite signals [4].

**3.4.1.1 Significance of Acquisition**

The general definition of acquisition is to identify visible satellites to the user (Receiver). If any satellites are visible or available to the receiver, acquisition process must determine the following properties of satellite generated signal. Those are explained in below sections.

**(a) Carrier Frequency**

The frequency of the signal from specific satellite is differing from its nominal value due to Doppler Effect. The Doppler frequency shift can occur when maximum velocity of the satellite is combined with a very high user velocity, leading to approach values as high as ±10 kHz of its nominal frequencies. For a stationary receiver on Earth, the Doppler frequency shift will never exceed ±5 kHz. After down conversion, the nominal baseband frequency (zero channel frequency) of the GLONASS signal on L1 corresponds to the IF.

**(b) Code Phase**

The code phase denotes the point in the current data block where the C/A code begins. It means the satellite generated PRN code (C/A code) and local generated C/A code matches from that point. If a data block of 1ms is examined, the data include an entire C/A code or beginning of a C/A code.

**3.4.1.2 Factors Effecting Acquisition Performance**

Several factors can affect the performance of acquisition process. One of the most important factors is the length of data used for acquisition process to acquire the code phase offset and Doppler shift frequency. Another factor is the frequency search step size in calculating the carrier frequency. More search steps will provide finer frequency at the cost of more processing time.

**(a) Data length for Acquisition**

It is very important to determine the length of data is required to perform acquisition. The longer the data the more is the processing time. The benefit of using a longer period of data is the improvement of the SNR (signal to noise ratio) of the signal.

The two factors that limit the selection of longer data period are the length of the navigation data and the change in Doppler frequency. Since the navigation data bit length is 20 ms, if the first 10 ms of the data has a phase transition due to the navigation data bit, there will be no phase transition in the next 10 ms of data. Therefore, the maximum data length that can be used is 10ms. Since the C/A code is one millisecond long, the beginning of a C/A coded input signal can be found within this amount of data. However, there is still a possibility that a phase transition occurs in this one millisecond of data due to the navigation data. If the transition occurs in this one millisecond of data, the next one millisecond of data cannot contain another navigation data bit transition. In order to guarantee that there is at least one set of data with no data bit transition, a minimum of two consecutive milliseconds of data should be used for successful acquisition. As noted previously, a maximum length of 10 ms of data can be ensured to contain no navigation data bit transition. Thus there is a compromise between the processing time and data length used for acquisition. Only one or two milliseconds of data are required for strong signals while the weaker signals need five to ten milliseconds of data for acquisition [9].

**(b) Doppler Frequency Search Step**

In the Acquisition process, Doppler frequency plays a vital role to determine the visible satellite. Doppler Effect occurs due to the relative motion of satellite and receiver (user), when the satellite and receiver are near to the each other then the Doppler frequency is very high and similarly when the receiver and satellite maintains a far distance between each other the Doppler frequency is very less. In Doppler removal process, the Doppler frequency is removed from the incoming carrier signal by mixing it with a locally generated signal. If the locally generated signal matches the incoming pseudo-base-band signal, the errors will be zero and the effect of Doppler frequency is removed.

Doppler frequency search step is another key factor in the process of successful and fast acquisition. Since the carrier frequency is affected by the Doppler shift, the shift has to be considered in the receiver generated signal. However, the exact Doppler shift is unknown, so the search needs to cover the range of all possible Doppler frequencies. The maximum Doppler frequency that needs to be searched is ±10 kHz which is the sum of both the Doppler on the C/A code and on the carrier frequency (IF). However, the number of frequency steps necessary to cover the 20 kHz frequency range needs to be determined. Widening the bandwidth of the searching steps improves the speed of the search. Using a narrow bandwidth in the search process requires more steps to cover the same desired frequency range resulting in more computation time. However, searching with narrow bandwidth steps improves the sensitivity.

### 3.4.2 Tracking

Once acquisition process completes, the acquisition gives rough estimation of the carrier frequency with Doppler and code phase offset parameters of all visible satellites of the receiver. These parameters are fed into tracking part. The main purpose of tracking is to refine these values, keep track, and demodulate the navigation data from the specific satellite. In total satellites, visible satellites are only required for tracking of satellites signals and non visible satellites are not required. The estimated parameters of all visible satellites from the acquisition i.e., carrier frequency, code phase are required to process tracking of a GLONASS L1 or L2 signals. In the tracking process of satellite signal, both carrier and code signal are accurately reproduced inside the receiver. In this process frequency error (phase error) and code phase occurs, and these two parameters can be minimized to zero and produce an exact carrier wave to demodulate navigation data of satellite signal. To generate an exact carrier wave, two tracking loops are required. Code tracking loop generates exact code phase of incoming signal similarly frequency or phase locked loop generates exact carrier wave replica to demodulate navigation data. This navigation data is required to compute the position of satellites, receivers.

## **3.5 Conclusion**

This chapter explained about the general view of GNSS Software receiver and each unit of in this receiver has been discussed. In the receiver front end RF to IF conversion process and each component in RF front end have been explained. However base band processing of software receiver consist acquisition, tracking which have been introduced and the acquisition and tracking algorithms will be discussed in next chapter.

# 4. GLONASS C/A code Acquisition

## 4.1 Introduction

In GLONASS Software Receiver, GLONASS C/A code acquisition is an important module. Its purpose is to identify the carrier frequency and code phase values in the incoming signal. This acquisition process can be done through various methods. Each method has some advantages and disadvantages. Suppose if the incoming signal is strong, it is better to choose fast-low sensitivity acquisition method, otherwise the FFT method is better to detect the signal. However, in case of weak signals, some other methods are presented. For the purpose of this work, only the cell - by - cell search method and FFT method are explained in next sections.

The Acquisition Process consist the following list

* Determine which satellites are visible to the antenna.
* Determine the approximate Doppler frequency of each satellite.
* Search for the signal in both C/A code phase and carrier frequency

## 4.2 Acquisition Methods

This section of Acquisition methods describes the GLONASS signal acquisition process for both conventional and software receiver architecture. Acquisition is a synchronization process giving estimates of the PRN code offset and the carrier Doppler. Most commonly used methods are:

1. Serial search in time domain
2. Cell – by – Cell search method
3. Parallel search in frequency domain or FFT method
4. Parallel frequency space search acquisition.
5. Parallel code phase search acquisition.

However serial search is the slowest search method, it is usually implemented in hardware based receivers due to its simplicity and low cost. The parallel search (FFT method) in frequency domain is usually implemented in software receiver since in serial search method computations are high. All acquisition methods are discussed in detail in the next section.

### 4.2.1 Serial Search in Time Domain

Serial search is the simplest and most frequently used acquisition method especially in hardware GLONASS receivers or GNSS receivers [6]. It can be directly implemented based on block diagram is shown in figure 4.1.

( ) 2

ʃ

PRNcode generator

Local oscillator

ʃ

Incoming signal

( ) 2

90°

Output

##### Figure 4.1: Serial Search Algorithm

In this algorithm initially digital IF or incoming signal is multiplied by locally generated PRN code sequence (C/A code) and locally generated carrier signals. The local C/A code generator generates a same C/A code for all GLONASS satellites. The generated sequence has a certain codephase, from 0 to 511 chips. When it is sampled with a sampling frequency of 16MHz, then length of one C/A code becomes 16000 samples, from this each sample takes 1 kHz frequency resolution.

Thus the incoming signal is initially multiplied by the locally generated PRN sequence, after this the signal is multiplied by a locally generated carrier signal to generate the in-phase signal (I), and multiplied by a 90o phase-shift version of locally generated carrier signal to generate the quadrature signal (Q).

The last end parts of the serial search algorithm are integration and squaring of the two results of the multiplications with the cosine and sine signals, respectively. The integration is simply a summation of all 16000 points corresponding to the length of the processed data (1ms). Integrator can also acts as low pass filter, it eliminates high frequency components. The squaring is then performed on the result of the summation to obtain the signal power. The final step is to add the two values from the *I-*arm and the *Q-* arm.

The output is a value of correlation between the incoming signal (digital IF) and the locally generated signals that can be done using cell –by – cell search method [2]. It is a two dimensional search process and is shown in figure 4.2 in which the x- axis represents the code phase and the y-axis represents the Doppler frequency bin, when both frequency and code phase are well aligned, then the correct alignment is identified by measurement of the output power of the correlators.



##### Figure 4.2: Cell – by – Cell Search method

In the cell – by – cell search method, the correlation is performed in the time domain, where the locally generated C/A code is shifted and accumulated for all possible shifts. Therefore, for GLONASS signal acquisition, all 511 chips of C/A code are examined. In this process code chip is searched in increments of half chip and each code phase value to be searched is considered a code bin. This process is repeated for all Doppler bins. Like this, the entire range of Doppler frequency search is divided into smaller cells called Doppler bins. One code bin and one Doppler bin create a cell.

Once cells are formed, then the codephase search starts from particular code bin and particular frequency bin. This search procedure gives a maximum value in that frequency bin and all code phase samples, that maximum value is stored in one table. After completion of search in all frequency bins, codephase the result of all maximum values are stored, in that result table the maximum of maximum values represents the beginning point of code phase value or first peak. Around this peak value the remaining samples are needed to find the second peak value. If the ratio of first and second peak exceeds the defined threshold value then the satellite is detected otherwise the signal search continues until acquisition is successful. Thus the acquisition space search can be performed by cell – by – cell search method in serial search algorithm.

The serial search in time domain algorithm performs two different sweeps:

1. Frequency sweeps over all possible carrier frequencies of IF ±7 kHz(in this case Doppler frequency is set to ± 7kHz ) in steps of 500 Hz
2. Code phase sweep over all 511 different code phases. All in all, this sums up to a total of

511 = 511× 29 = 14819 combinations

Where 511 represents code phase, remaining are represents frequencies

Obviously, this is a very large number of combinations. This exhausting search procedure also tends to be the main weakness of the serial search acquisition.

Usually when serial search is performed the code phase is incremented by chip when one code phase has been tested. Therefore, there are 2×511 = 1022 code phases that must be searched. In the software receiver implementation, with the input signal is sampled at 16MHz, so 32,000 samples of the received data are correlated with the locally generated C/A code by sliding the replicated code over the 32,000 samples. The frequency resolution obtained from the 1ms of data is about 1 kHz; the frequency resolution is determined by the coherent integration time (or dwell time). The relation is

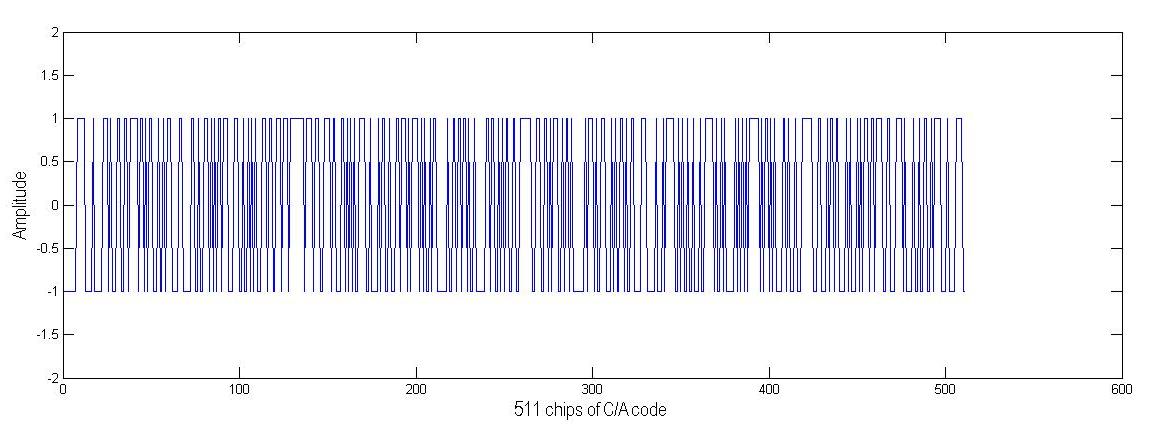
D =

Where D is the frequency bin width in Hz and T is the predetection integration time in seconds.

By stripping the C/A code from the input data, the remaining signal becomes a continuous wave. Once the input signal becomes a continuous signal, Fast Fourier transform (FFT) can be used to find the frequency, and this operation is referred to as coherent integration. It processes only 1ms of input data. However a set of long input data is divided into many blocks and coherent integration is performed on all the blocks. After the coherent integration every frequency output is complex and can be put into amplitude form. The amplitudes from the all coherent integration of the same frequency are summed. Non coherent integration uses the outputs of coherent integration. From this it can be concluded that serial search algorithm is not useful in software receiver approach, this can be used in conventional hardware receivers (or ASIC) due to its low cost and complexity.

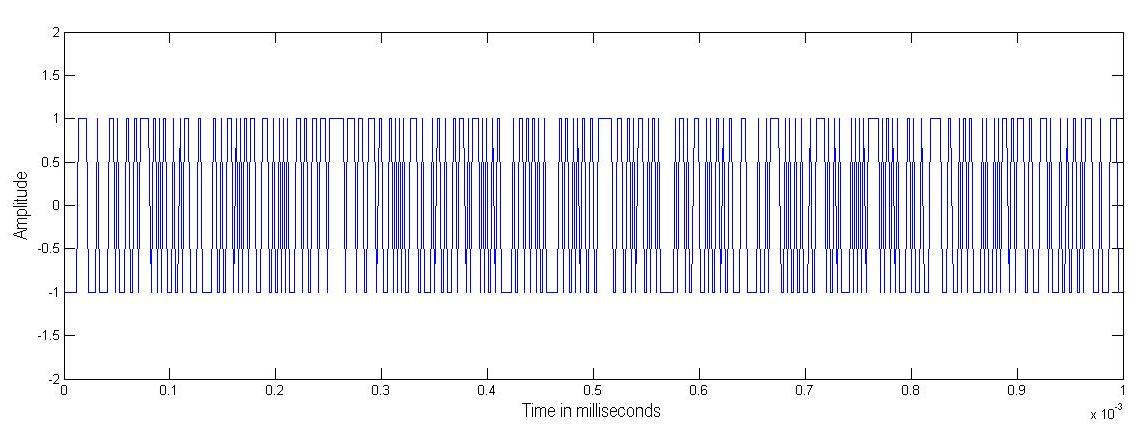
**(a) PRN Sequence (C/A code) Generation**

The main task in any acquisition method is to multiply the incoming signal with the locally generated PRN sequence. The generated PRN (Pseudo Random Noise) code is a series of bits which are produced by using a 9 bit shift register operated by 0.511MHz clock, it is produced using the polynomial equation 1+ x5 + x9. In the GLONASS case, generated C/A code is having a period of 511 chips which are repeated for every 1ms. The generated 511 chips code from linear feedback shift register is shown in below Figure 4.3.



##### Figure 4.3: 511 chips of C/A code

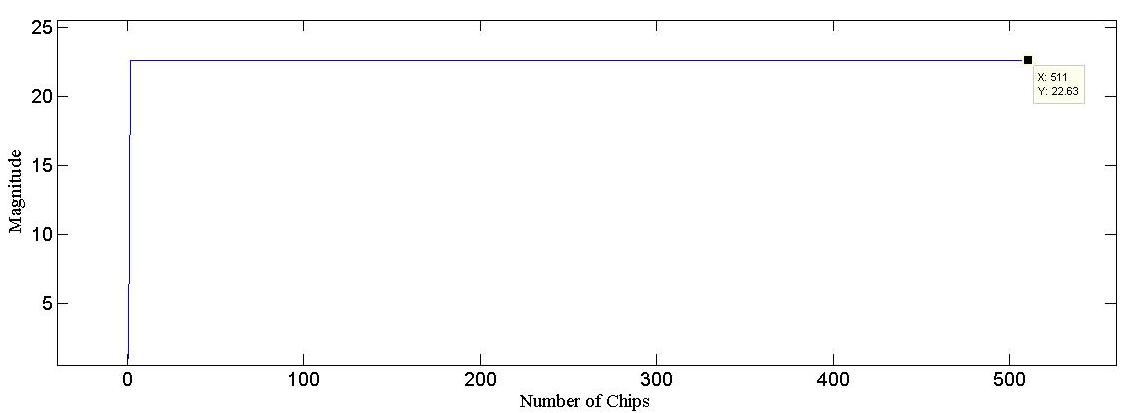
The generated C/A code is having duration of 1ms, which consist one set of C/A code at sampling frequency of 0.511MHz shown in below via by expressing C/A code vs. Time × (0: code length -1)



##### Figure 4.4: C/A code vs. Time

To the above C/A code, when absolute FFT or DFT is applied, it will transform the C/A code from time domain to frequency domain. The magnitude of all sample having the same value that has been shown in below figure.

× (0: C/A code length -1) = × (0:511-1)

****

##### Figure 4.5: Frequency domain representation of CA code

**(b) C/A Code Generations for Different Sampling Frequencies**

Sampling frequency can be defined as a discrete time signal generated from a continuous signal. Therefore the discrete signal preserves the same information without changing their properties. If C/A code is sampled at 2.044MHz means C/A code sampled to 4 times of its code frequency so here each chip will be replaced with 4 samples. The spectrum of 2.044MHz sampled C/A code is shown in below Figure4.6.

Fs ≥ 2Fmax = 2.044MHz ≥ 4×0.511MHz

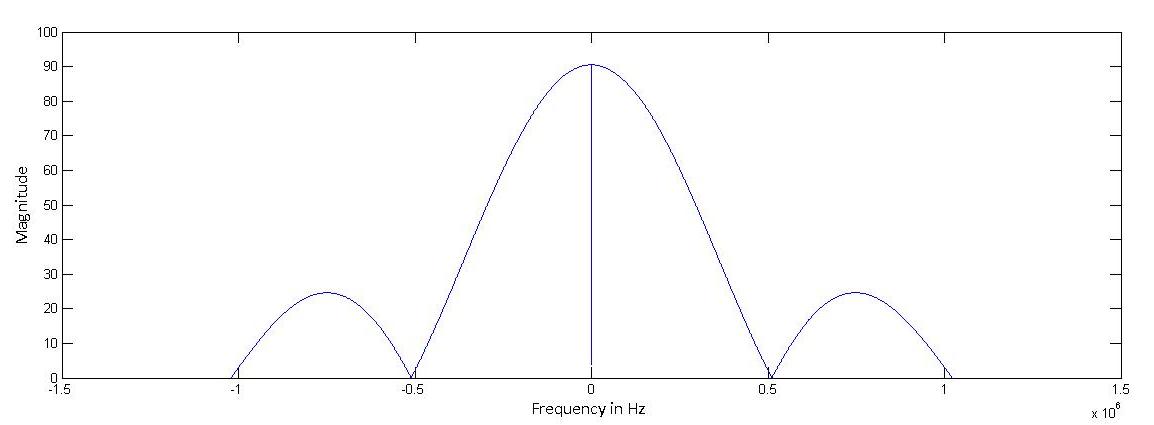


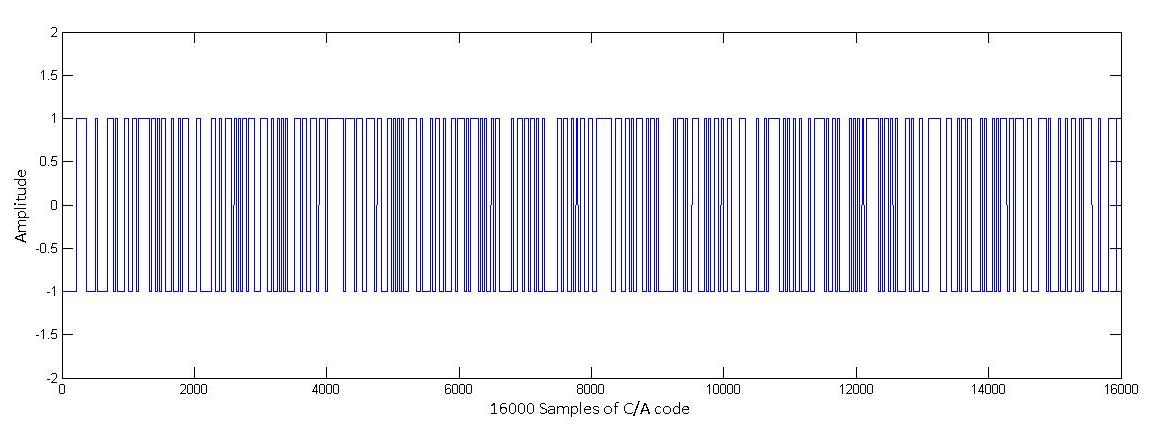
Figure 4.6: Spectrum of C/A code sampled at 2.044MHz

In the Figure 4.6, graph indicates that the minimum frequency is 2.044MHz and the maximum frequency is 0.511MHz. In GLONASS receiver, sampling of C/A code plays an important role while digitizing IF signal and removing the C/A code in the digitized incoming signal at receiver side.

Sampling frequency (Fs) = 32 × 0.511MHz = 16MHz

Fmax = = 0.511MHz.

In this work when 0.511MHz C/A code is sampled to16MHz sampling frequency, so each chip will be sampled or digitized by 31 samples and the frequency resolution of each sample takes 1 kHz. Therefore 511 chips will be sampled to16000 samples/ms, the 16MHz sampled C/A code is shown in below Figure 4.7.

****

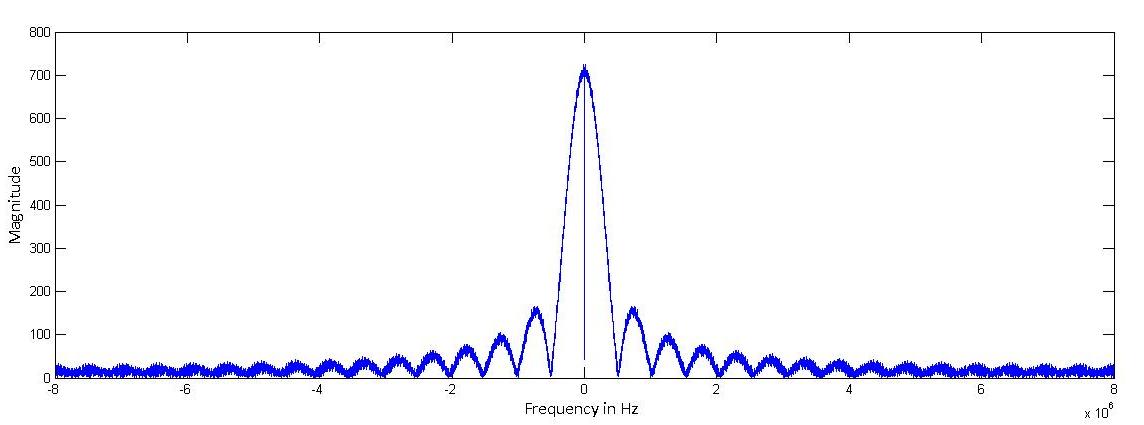
##### Figure 4.7: 16000 samples of C/A code

After sampling C/A code of 0.511MHz to 16MHz,

Sampling time (Ts) = 6.25×10-8sec

Chip duration (Tc) of 0.511MHz C/A code = 1.95×10-6sec

To the above C/A code, when DFT or FFT is appliedto 16MHz sampled C/A code, then the spectrum of C/A code is as shown in Figure 4.8. The transmitted C/A code is having main lobe and several side lobes because sampling frequency is very high compared to code frequency. The main lobe of spectrum is having 1.022 MHz bandwidth, and the maximum frequency value occurs at 0.511MHz because this is clock frequency of C/A code.



##### Figure 4.8: Spectrum of C/A code sampled at 16MHz

**(c) Carrier Wave Generation**

The second step in acquisition is multiplication with a locally generated carrier wave. The same carrier can be generated locally at the GLONASS Receiver using local oscillator. The local carrier generator must generate two carrier signals with a phase difference of 0° and 90°, corresponding to a cosine and a sine wave. The carrier must have a frequency corresponding to the IF ± the Doppler frequency step according to the examined frequency area. It must be sampled with the sampling frequency of 16MHz. Thus a complex signal is generated using the natural exponential function *exp (j2πf)* and these generated carrier waves are multiplied with satellite generated signal so that the carrier signal can be removed. However in the GLONASS system the carrier frequencies of L1 and L2 sub bands are allocated to channels as shown in Table 2.2.

### 4.2.2 Parallel Search in Frequency Domain

The serial search acquisition method is a time-consuming procedure to search sequentially through all possible values of the two parameters i.e. carrier with Doppler frequency and code phase. If any of the two parameters could be eliminated from the search procedure or if possible implemented in parallel, the performance of this procedure increases significantly. This second method of acquisition parallelizes the search for one parameter which can be frequency or code phase. This method utilizes the Fourier transform, to perform a transformation from the time domain into the frequency domain. This is also called FFT search method [2] [6].

There are two standard methods in this parallel search acquisition:

* Parallel frequency space search acquisition
* Parallel code phase search acquisition

**4.2.2.1 Parallel Frequency Space Search Acquisition**

In this method of acquisition, parallelizing the search for frequency, so it has been named Parallel Frequency Space Search. Initially this method is identical to the serial search method. The parallel frequency space search algorithm can be implemented directly based on the block diagram shown in Figure 4.9.

Incoming signal

Fourier transform

PRNcode generator

││2

Output

##### Figure 4.9: Parallel frequency space search algorithm

Initially in the parallel acquisition process, the digital IF (incoming signal) is multiplied by a locally generated PRN sequence, with a codephase corresponding to a specific satellite having 0 to 511 chips, after the code multiplication, resulting signal is transformed into the frequency domain through a Fourier transform. The Fourier transform could be implemented as a Discrete Fourier Transform (DFT) or a Fast Fourier Transform (FFT). If the locally generated code is well aligned with the code in the incoming signal, the output from the FFT will have a peak at the IF plus Doppler offset frequency. The absolute values of all components are calculated in order to determine the frequency of a possible peak. The peak will be located at the frequency index. It corresponds to the frequency of the carrier wave signal. However the parallel frequency space search acquisition method searches or steps 511 different code phases. Therefore the implementation in parallel frequency domain makes faster than the serial search method.

In this method, if 10ms of data is analyzed, the number of samples can be found as of the sampling frequency. That is, if the sampling frequency is Fs= 16 MHz, the number of samples is *N* = 160000.

With a DFT length of 1, 60,000, the first *N/*2 output samples represent the frequencies from 0 to Hz. That is, the frequency resolution of the output is

∆f = = ………………………………………………..…Eq 4.1

With a sampling frequency of *Fs* = 16 MHz the resulting frequency resolution is

∆f = = 100 Hz

Here the accuracy of the frequency depends on length of the DFT or the number of samples in data analyzed. The FFT is the faster of the two; but it requires an input sequence with a radix-2 length, that is, 2n, where *n* takes positive integer value.

The parallel frequency search method eliminates the search through the possible (29 frequencies) different Doppler frequencies. Generally this algorithm is used to fine the resolution of estimated carrier frequency of incoming signal. It is possible only when the estimated codephase is available from coarse acquisition by using parallel code phase search method. The searching in all 29 possible frequencies can be done through the Parallel Code Phase Search Acquisition by parallelizing the code phase.

**4.2.2.2 Parallel Code Phase Search Acquisition**

Another standard method of acquisition is parallelizing the codephase search, so it is named as Parallel codephase searches. Parallel frequency search method eliminates the search through the possible different frequencies, but here it eliminates the search through all possible code phases. In the same way as with the other acquisition methods, it is directly based on the block diagram shown in Figure 4.10.

Here initially the incoming signal is multiplied with a generated cosine and sine carrier wave from the local oscillator, obtaining an Inphase (I) and a Qudrature (Q) signal to be used as a real and imaginary, these two are combined as a complex input to the Fast Fourier transform or DFT function. The result is multiplied by the DFT of complex conjugate of a specific PRN code.

After the multiplication of DFT result and complex conjugate of PRN code, the resulting sequence is fed into input to the IDFT; it will transform from frequency domain to time domain. Then it finds the magnitude of the resulting sequence which is calculated by using circular correlation method.

Fourier transform

Fourier transform

Inv Fourier transform

││2

Local Oscillator

PRN code generator

Complex Conjugate

90°

Incoming signal

Output

Q

I

##### Figure 4.10: Parallel code phase search algorithm

The main goal of this acquisition is to perform a circular correlation with an incoming signal and a locally generated PRN code. The goal of this method is to parallelize the code phase search which greatly reduces acquisition times compared to previous search methods. The amount of search steps in the code phase dimension is significantly larger than that in frequency (Instead of using 511codephases here we are using only 29 frequencies). When the codephase is implemented in parallel, only 29 steps of frequencies are required which is very less when compared to 511 in the parallel frequency space search acquisition algorithm. So this method can simply referred to as parallel code phase search acquisition algorithm.

The result of this algorithm is very similar to that obtained using the serial search. The maximum value of the resulting sequence corresponds to the best estimate of the code phase of that PRN sequence in the data set (1ms) for the frequency bin index tested. If that maximum value exceeds a predetermined threshold, a peak is present in the correlation; the index of this peak marks the beginning of PRN code phase of the incoming signal. If that maximum value does not exceed the predetermined threshold, then signal is not present. Thus the remaining parameters, PRN codes and frequencies, can be correlated, until acquisition is successful.

Comparing all the previous acquisition methods, the parallel code phase search acquisition method has cut down the search space to only 29 possible different carrier Doppler frequencies. For each satellite acquisition, the Fourier transform of the generated PRN code must be performed once. For each of satellite 29 frequencies should perform one Fourier transform and one inverse Fourier transform, so the computational efficiency or performance depends on the implementation of these functions.

###### Table 4.1: Comparison of Acquisition algorithms

|  |  |  |
| --- | --- | --- |
| Algorithm | Repetitions | Complexity |
| Serial Search | 14819 | High |
| Parallel Frequency Search | 511 | Medium |
| Parallel Codephase Search | 29 | Low |

(1) Compared with serial acquisition method, the FFT based method changes two-dimension search of code delay and Doppler shift to one-dimension search, improves the acquisition speed and performance.

(2) FFT-based acquisition method avoids hardware cost by only software modification and little hardware modification and with good flexibility.

(3) FFT algorithm is applicable for high speed digital signal processor and can meet real-time and accuracy demand of full digital receiver.

Therefore the FFT search algorithm is used to estimate codephase and frequency parameters in MATLAB implementation because of its significant fast acquisition speed compared to serial search method.

The PRN codephase accuracy estimated using this acquisition method is more accurate compared to the other methods. If sampling frequency is 16MHz, a sampled PRN code has 16,000 samples in 1ms, so the accuracy of the code phase can have 16,000 different values instead of 511.

The proposed acquisition algorithm uses circular correlation which is different from convolution. A correlation performed between x (n) and h (n) having same length ‘N’ (16000) can be written as

R (n) = ……………………….…….…………Eq 4.2

Where: x (m) = input signal

h (n)= it is not impulse response of the system but that of another signal

If the DFT performed on Eq 4.2, the result is

R (k) =

=

=

= ……………………………………………...Eq 4.3

Where: X-1(k) represents the inverse DFT.

The above equation 4.3 can also written as

R (k) =

=

If the x (n) is real, x (n)\* = x (n) where \* is the complex conjugate using this relation the magnitude of R (k) can be written as

………...…………………….Eq 4.4

The relationship can be used to find the correlation of the input signal and locally generated signal. The above equation provides a periodic (circular) correlation and this is the most desired procedure.

**Acquisition using Circular correlation**

The acquisition process of the software receiver uses the circular correlation method to find the signal. The parallel code phase search algorithm can be implemented using circular correlation [9]. The circular correlation is a multiplication in the frequency domain that can be expressed as.

R [m] = x (n) CA [-n] = F-1(F(x[n]).F (CA[n]\*))………………………….…Eq 4.5

Circular correlation

Where F is the fast Fourier transforms (FFT) and F-1 is the inverse (IFFT) is used to calculate *R.* Since the fast Fourier transform is used to implement the DFT and IDFT, the acquisition is also called the FFT search.

In this correlation process, initially 1ms of input data is used to obtain the beginning point of the C/A code in the incoming signal. Since the sampling frequency of the incoming signal is 16MHz, so there are 16000 samples in 1ms duration. The C/A code does not necessarily start at the beginning of the input signal therefore the beginning of the C/A code in the incoming signal is needed before despreading the data. A local copy of the specific C/A code is generated for 1ms and sampled at a frequency of 16 MHz to create the 16,000 samples needed. The sampled C/A code is then slid over the 16,000 sample input data to find the peak correlation value. This peak value represents beginning point of C/A code, this estimation of codephase is utilized in fine frequency resolution algorithm.

**Fine Carrier Frequency Resolution**

Based on the accurate estimated code phase from parallel code phase search acquisition, the continuous signal can be obtained by stripping the C/A-code from the incoming signal. After applying FFT to obtained continuous signal, the result gives highest power of frequency index value. It represents obtained estimated carrier frequency.

In fine frequency resolution to find the estimated carrier frequency with finer resolution, the FFT is an efficient method. The Fourier transform can be implemented by DFT or FFT. FFT is faster but it always requires the input sequence with a length of 2n, where *n* is a positive integer. The FFT algorithm cannot be applied directly. The most often used method is to add 0’s to the input sequence. This method does not affect Fourier transform result of sequence.

To obtain 100 Hz frequency resolution with FFT, the data length required is 10ms.With the sampling rate of 16 MHz, a total of 1, 60,000 points FFT has to be computed. Similarly to obtain 10Hz frequency resolution with FFT, the data length required is 100ms, which is a time consuming operation.

## 4.3 Algorithm Implementation

The proposed frequency domain parallel acquisition method can be performed by the following two steps [4]:

The first step is the coarse acquisition, which utilizes the parallel code phase search acquisition method. It is in steps of 1 kHz over possible frequencies of intermediate frequency IF±7kHz. In coarse acquisition it requires two DFT, one N point complex multiplications and one IDFT. After performing the frequency domain correlation in which the DFT is implemented by mixed-radix FFT, the accurate C/A-code phase is obtained and the carrier frequency is found with 1 kHz resolution which is too coarse for further signal tracking. If the ratio of the first two maximum peaks after the frequency domain correlation is above the preset threshold, enter fine acquisition to look for the more accurate carrier frequency; otherwise return to the beginning of the acquisition.

The next step is the fine acquisition, which utilities the parallel frequency space acquisition method. Based on the accurate estimated code phase from previous coarse acquisition step, the continuous signal can be obtained by stripping the C/A-code from the incoming signal. The more accurate carrier frequency is obtained via a FFT approach as well as increasing the length of the FFT [4].

### 4.3.1 Coarse Acquisition

1. Perform the DFT on the locally generated C/A-code *c* (*n*) to convert it into frequency domain as *C* (*k*). Take the complex conjugate *C* (*k*) and the output becomes *C*\* (*k*).
2. Generate the local carrier signal S*Ii (n)* and its 90°phase-shifted version S*Qi (n)*.
3. Multiply the 1ms incoming signal *x*1(*n*) by S*Ii (n)* and S*Qi (n)* respectively, giving an I and a Q signal component. The combination of these two generates a complex input to the DFT function and calls the acquisition result1 *X1i (k)*. Do the same operation on the following 1ms incoming signal *x2*(*n*) and call the acquisition result2 *X2i (k)*.
4. Multiply *X1i (k)* and *X2i (k)* by *C*\* (*k*) respectively and call the results *R1i (k)* and *R2i (k)*;
5. Take the inverse DFT of *R1i (k)* and *R2i (k)* to transform the results into time domain and find the absolute values as |*r1i (k)|* and |*r2i (k)|*, among which take the one where the maximum correlation peak is as the correlation result |*r1i (k)|* and record it.
6. If all the frequency bins are tested, we get *r* (*n*) composed of all |*ri (k)|* s with different frequency bins and then go to step g, otherwise go to step b.
7. The maximum of *r* (*n*) is the desired result, if the ratio of the first two maximums of *r*(*n*) is above the preset threshold. The C/A-code phase T is marked corresponding to the index of the peak.

### 4.3.2 Fine Frequency Acquisition

1. Strip the C/A-code from the 10ms incoming signal using the estimated C/A-code phase from previous coarse acquisition and then obtain the 10ms continuous signal as *x c* (*n*);
2. Find the positive integer *n* satisfying the inequality 2*n* *length* (*k.x c* (*n*)) and take *M* = 2*n* as the length of the FFT, where *k* is a positive integer;
3. Take the *M* -point FFT of *x c* (*n*) to transform it into frequency domain and find the absolute value as |*XC*(*k*)|.
4. Find the maximum of |*X C* (*k*)| and calculate the refined carrier frequency according to the index of the peak.

## 4.4 Observations of Acquisition algorithm

To verify acquisition process properly, first it uses 1ms of incoming signal (data block of 1ms) similarly it takes another 1ms of incoming signal, these two signals of 1ms duration performs correlation with a locally generated C/A code signal to estimate the beginning point of codephase value of acquisition search satellites.

From the Figure 4.11, it illustrates the result of acquisition process for simulated GLONASS satellite (PRN) 3, where the peak represents estimated beginning point of C/A code and carrier frequency of satellite 3 calculated by the acquisition program. In the 16000 point sampled input data of 1ms duration, the peak value of satellite 3 occurs at sample of 6999 means at this point, the incoming signal and local generated C/A code signal matches, also frequency of that satellite matches at below value.

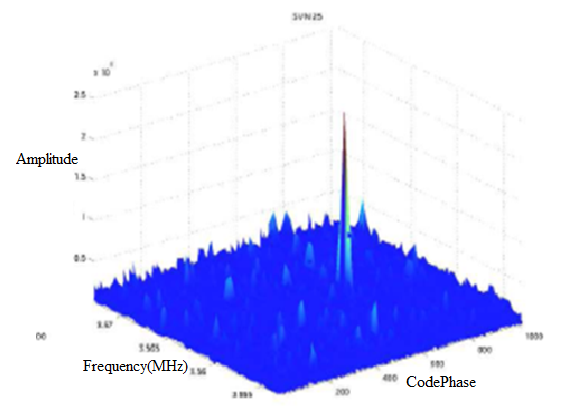


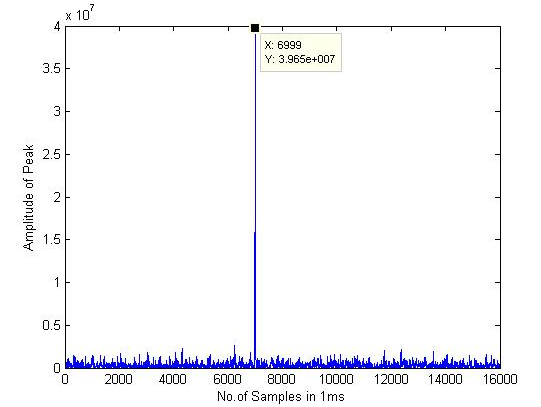
Figure 4.11: Acquisition of Satellite 3 at IF with Doppler frequency bin number 16

(IF + 3×0.5625MHz) – × 1000 + 0.5 kHz × (16 - 1) = -1.812MHz

Where IF = -3.5MHz

Above frequency (-1.812MHz) value consist highest frequency component, the correlation peak at this frequency component is much greater than acquisition threshold value. So it is the estimated carrier frequency with Doppler shifts of satellite.

From the Figure 4.12, it illustrates correlation between the incoming signal and locally generated C/A code signal, while performing correlation it is found that at 6999 sample the beginning of satellite 3 and incoming signal are well aligned.



##### Figure 4.12: C/A code begins at sample 6999 for GLONASS satellite 3

In the Figure 4.13, 10ms of data signal consisting 1, 60,000 samples each having 100Hz frequency resolution; carrier frequency of particular satellite from estimated codephase value of coarse acquisition can be obtained from parallel frequency search acquisition.

From the Figure 4.13, it can be verified that the Doppler shift did not cause the ideal carrier frequency to shift much. The initial estimated carrier frequency is still 1.812MHz; however the final carrier frequency output of the acquisition program is 1.811890MHz this value is obtained after several frequencies refined searches. Since the initial frequency resolution is 1 kHz, the estimated carrier frequency will be too coarse for the tracking program. The bandwidths of the tracking loops are very narrow, so the final estimated carrier frequency should be within a few Hertz of the actual carrier frequency.

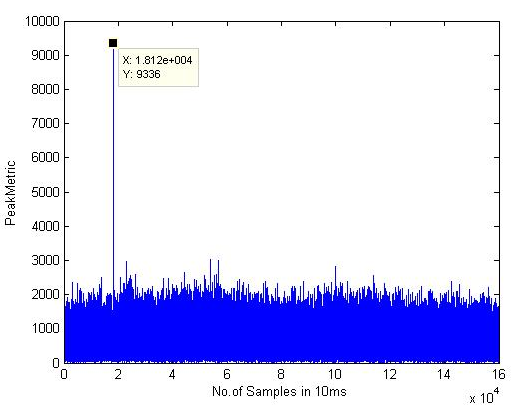


Figure 4.13: Acquisition of satellite at IF 1.812MHz

Therefore similarly for all visible satellites, carrier frequency can be estimated from codephase value of coarse acquisition algorithm, by using codephase value C/A code from incoming signal can be removed.

Suppose after performing acquisition algorithm such as parallel frequency search method or serial search method, if no satellite is available in data signal (i.e, it’s peak doesn’t cross predetermined threshold value) satellite is not visible. The Figure4.14 shows no satellites signal.

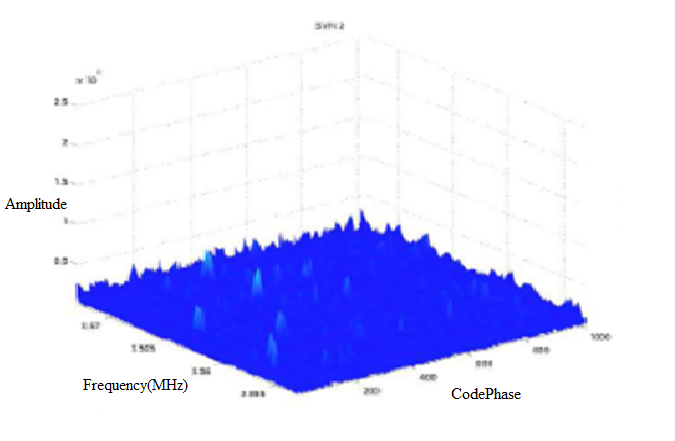


Figure 4.14: No satellite found for GLONASS satellite PRN 2

### 4.4.1 Results from Acquisition

The acquisition algorithm is applied to the collected data for all possible 14 satellites in which only the first 10 milliseconds of data have been utilized. In this,10ms of data signal (in coming signal) is correlated with locally generated signals when both are well aligned, it results in a codephase value; by using this, carrier frequency can be identified. However while doing acquisition, six different GLONASS satellites are identified in the collected data. The estimated parameters of identified satellites in acquisition are presented in Table 4.2.

###### Table 4.2: Visible satellites parameters

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Channel | PRN | Frequency(in Hz) | Doppler(in Hz) | Code offset |
| 1 | 10 | 2123581 | -1419 | 7545 |
| 2 | 3 | -1811890 | 610 | 6999 |
| 3 | 4 | -1245384 | 4616 | 3750 |
| 4 | 9 | 1563103 | 693 | 2117 |
| 5 | -- | -- | -- | -- |
| 6 | -- | -- | -- | -- |
| 7 | -- | -- | -- | -- |

## 4.5 Conclusion

In this chapter, several related GLONASS signal acquisition algorithms are introduced and the comparison between the conventional time domain serial search and frequency domain parallel acquisition method is discussed. In the process of GLONASS signal acquisition, the frequency domain parallel acquisition algorithm based on FFT approach is used. It will estimate the beginning point of C/A code in incoming signal and carrier frequency obtained by fine frequency resolution with a resolution of 100Hz. The data length for acquisition is to be chosen based on the desired sensitivity to fasten the results of averaging methods, which improves sensitivity of the receiver and also performs the computation effectively. Thus the acquisition is performed to estimate code phase and carrier frequency, which are needed to track the incoming signal. Tracking will be discussed in next chapter.

# 5. Tracking

## 5.1 Introduction

The acquisition process gives rough estimation of the frequency and code phase parameters, which are given as inputs to tracking loop. The function of tracking loop is to track variations in the incoming signal and to demodulate navigation data in the incoming signal. The demodulated navigation data bits are utilized in the navigation data decoding to find the position of receiver on the earth.

## 5.2 Tracking

The main goal of tracking is to synchronize a locally generated PRN code (C/A code) or replica of the transmitted PRN code with the incoming signal. For this purpose, closed tracking loops are used to continuously track the codephase and carrier frequency of the incoming signal. These loops are often referred as code and carrier tracking loops. This simple process has two functions:

**•** Demodulates the navigation data from the incoming satellite signal

**•** Givesthe frequency and phase errors between the reference and input signals.

The two standard types of tracking loops being used are:

1. Delay Locked Loop (DLL)

It can also be called as Code Tracking loop or Delay locked loop

1. Carrier Locked Loop (CLL)

It can also be implemented by Frequency locked loop (FLL), Phase locked loop (PLL).

The type of tracking loop to be used is decided by the parameter of signal. When phase of the signal is considered phase locked loop (PLL) can be used similarly when frequency of the signal is considered frequency locked loop can be used. Hence these tracking loops play an important role in refining the carrier and codephase of the signal.

### 5.2.1 Basic Demodulation of satellite signal

Let FL1 and FL2 be the carrier frequencies of L1 and L2 sub bands of the signal transmitted from satellite ‘k’ with powers PC, PPL1, and PPL2 for C/A or P code [6]. The C/A code sequence is Ck (t) and the P(Y) code sequence is Pk (t). If the navigation data sequence is Dk (t), then the incoming signal to the demodulator can be mathematically represented as

Sk (t) = Ck (t) Dk (t) cos (2π FL1t) + Pk (t) Dk (t) sin (2π FL1t) + Pk (t) Dk (t) Sin (2π FL2t)....................................... Eq 5.1

Navigation data

Local Carrier

Local code

Incoming signal

##### Figure 5.1: Basic demodulation scheme to generate navigation data

The output from the front end including filtering and down conversion can be described as

= + …………….Eq 5.2

Where ωIF is the intermediate frequency, which is down converted carrier frequency of incoming signal at the front end.

This signal is sampled by specific sampling frequency (16MHz) using an A/D converter. Because of the narrowband filtering around the C/A code, the P-code is distorted. The last term of the equation can be filtered out and cannot be demodulated and is assumed as noise e(n). The signal from satellite ‘k’ after the A/D conversion can be expressed as

……….……………………….…Eq 5.3 ‘n’ indicates that the signal is discrete in time.

To obtain the navigation data Dk (n) from the above signal, the signal has to be down converted to baseband. In this process the carrier removal is done by multiplying the input signal with a replica of the carrier as shown in Figure 5.1. If the carrier replica is identical to the incoming signal carrier in both frequency and phase, the product of both is

= ……..Eq 5.4

Where the first term is the navigation message multiplied by the PRN code and the second term is a carrier of frequency which is twice the intermediate frequency. The signal after passing through low pass filter is

.........................................................................................Eq 5.5

The next step is to remove the C/A code Ck (n) from the signal. This is done by correlating the signal with a locally generated replica of code. If the code replica is exactly the same as the code in the signal, the output of the correlation is

……………………………………………..Eq 5.6

N Dk (n) is the navigation message multiplied by the N number of points in the signal.

## 5.3 Carrier Tracking Loop

To demodulate the data signal successfully, an exact replica of carrier wave is to be generated by using carrier tracking loop. The carrier tracking loop is also called carrier lock loop (CLL) and it must track the phase of the incoming carrier by using a phase lock loop (PLL) or the frequency of the carrier by using frequency lock loop (FLL). Therefore Carrier lock loop can be implemented by two different loops those are:

* Phase locked loop
* Frequency locked loop

### 5.3.1 Basic Phase locked loop

The basic block diagram of a phase locked loop is shown in Figure 5.2. In the first stage the input signal is compared with a locally generated reference signal to obtain the phase difference between the two. This difference is then filtered and applied as input to a voltage or numerically controlled oscillator. Based on the error signal, the oscillator adjusts the frequency of its reference signal to match with the phase of the input.

Phase Detector

Loop Filter

VCO

Figure 5.2: Simplified Tracking loop

The main purpose of a PLL is to adjust the frequency or phase of a locally generated signal inorder to match with the frequency or phase of an input signal, referred to as the reference signal. The basic phase locked loop in time domain is shown in Figure 5.3(a), which includes the transfer function, the error transfer function, and the noise bandwidth.

VCO

VC (t)

VO (t)

\_

+

ԑ (t)

k0

Low pass Filter

##### Figure 5.3(a): Time domain

ԑ (s)

**+**

**\_**

VCO

VC (s)

VO (s)

k0

Low pass Filter

##### Figure 5.3(b): S-Domain

Figure 5.3(a) shows the time domain configuration and Figure 5.3(b) shows the S domain configuration, which are related by the Laplace transform. The input signal or reference signal is   
and the output from the voltage controlled oscillator (VCO) is

In the process of tracking, carrier frequency is varied according to the incoming signal so that local oscillator is replaced by VCO to generate the same carrier. The applied input voltage to VCO determines the instantaneous oscillation frequency, it means when the input voltage changes automatically the output of VCO frequency changes. The phase comparator measures the phase difference between these two signals. The parameter ‘ko’ of the amplifier represents the gain of the phase comparator and the low-pass filter limits the noise in the loop. The input voltage ‘V0’ to the VCO controls its output frequency, which can be expressed as

Where ω0 is the centre angular frequency of the VCO and ‘k1’ is the gain of the VCO. The phase angle of the VCO can be obtained by integrating above Equation

Assume

The Laplace Transform of

*……….*……………………….………...Eq 5.7

From figure (b) the equation can be written

…………………………..….…Eq 5.8

From these three equations, one can obtain

(Or)

Where is the error function

The transfer function H(s)of the loop is defined as

…………………………………………Eq 5.9

where ‘k1’ is the discriminator gain, ‘k0’ is the NCO gain, and the F(s) is the transfer function of the loop filter. Combining the k0 and k1 constants into a single constant K results in the transfer function given by Equation (5.4)

……………………………………........Eq 5.10

The filter (F(s)) can provides following transfer functions

……………………………………………...…Eq 5.11

Substituting Eq 5.11 into Eq 5.10 then the final phase transfer function becomes

………………………………………....Eq5.12

Where natural frequency ωn = and Damping ratio ζ =

It is a second order tracking loop since the denominator of H(s) is a second order in ‘s’. The loop filter, F(s), is very important to a PLL. It determines the loop order, natural frequency, damping coefficient, and noise bandwidth, and thus greatly affects the overall performance. A first order loop is not used for tracking GNSS signals since the PLL steady state error due to a frequency step or frequency ramp (i.e., second or third derivative of the phase angle) does not go to zero due to the presence of only one integrator. In order to implement this loop in software, the continuous system must be changed into a discrete system.

Output

Incoming signal

\_

+

N (Z)

F (Z)

##### Figure 5.4: linearized digital second-order PLL

The linearized digital second-order PLL model is shown in above Figure, where ‘Kd’ is the discriminator gain, F(z) is the transfer function of the filter, and N(z) is the transfer function of NCO. The transfer from continuous ‘S’ domain into discrete ‘z’ domain is accomplished through bilinear transform,

s = where ‘ts’ is the sampling interval

The transfer functions of the digital filter and NCO and the loop filter transfer function in the ‘z’ domain is

And

Where ; and K = K0Kd

Where KoKd is the loop gain, ‘ζ’ is the damping ratio, ωn is the natural frequency, and ts is the sampling time.

The natural frequency can be defined as

Where: ‘BL’ is the noise bandwidth in the loop.

Output

C2

C2

C111

Z-1

Incoming signal

##### Figure 5.5: Second order phase lock loop filter

The selection of the natural frequency of the loop is a compromise. A small natural frequency will provide excellent noise performance but it will be unable to track dynamic variations of the signal. However a large natural frequency will be able to track signal but will have poor noise performance. These conclusions are based strongly on the noise bandwidth approximation of the tracking loop given as

………………………………...….Eq5.13

Based on the dynamics expected, typical noise bandwidths for the code and carrier tracking loops are 1Hz and 25Hz respectively. For a damping ratio of 0.707 these corresponds to a natural frequency of 2Hz and 50Hz respectively for the code and carrier tracking loops. This provides the most accurate discrete implementation of the linear phase lock loop model.

**Damping ratio**

The damping ratio controls how fast the filter reaches its settling point. It also controls the overshoot of the filter. A smaller settling time results in a larger overshoot. A larger settling time results in a smaller overshoot of the phase. This is shown in Figure5.6. The damping ratio plays an important role in the dynamic performance of the tracking loop. When working with an under damped (0 < ζ < 1) system the step response is rapid and will oscillate before settling down to the desired state. If the system is over damped (ζ >1), the step response will be slow in achieving the desired state but will do so without any oscillation. The optimally flat-response is achieved using ζ = 0.707, which corresponds to a second order Butterworth low pass filter. Thus the choice of damping ratio is a compromise between overshoot and settling time. The damping ratio is chosen to be ζ = 0.7 resulting in a filter that converges reasonably fast and does not make a high overshoot [6].



##### Figure 5.6: Different loop responses depending on damping factor

**Noise Bandwidth**

The second parameter in the PLL filter is the noise bandwidth ‘BL’.It controls the amount of noise allowed in the filter. As the tracking loop starts to track a signal the start frequency is the frequency found by the acquisition algorithm. The start frequency from the acquisition algorithm can be off by some Hz. The tracking loop is then going to lock onto the correct frequency. To see the impact of various noise bandwidths, a real GLONASS signal is used where the acquisition algorithm found a frequency that is about 25 Hz off [6].

### 5.3.2 Costas loop

In tracking process, carrier tracking can be done by phase locked loop. This loop can be implemented by using costas loop because it is insensitive to180°. The Costas loop [6] in Figure5.7 contains two multiplications.

The first is the product between the data modulated carrier signal and the local carrier wave and the second multiplication is between a 90◦ phase-shifted carrier wave and the data modulated carrier signal, which are then filtered using lowpass filter. The loop discriminator (arctangent discriminator is optimal) block is used to find the phase error on the local carrier wave replica. The output of the discriminator is then filtered by using loop filter and used as a feedback to the numerically controlled oscillator (NCO), which adjusts the frequency of the local carrier wave. In this way the local carrier wave could be generated as replica of the input signal carrier wave.

Carrier Loop Filter

Low Pass Filter

90°

Carrier Loop Discriminator

Local Carrier Generator

Low Pass Filter

PRN code

Incoming Signal

##### Figure 5.7: Costas loop block diagram

Assuming the loop is operating in lock operation, the modulated data bits are present on the in-phase arm of the costas loop, directly after the lowpass filter. In order to simplify the analysis, a linear model of the feedback tracking loop shown in Figure 5.4 is often used. It has been shown that this linearization can be used to model the Costas loop implementation of Figure 5.3. This linear model is depicted in Figure 5.3(b) with the phase transfer function given in Equation 5.12. The goal of the Costas loop is to try to keep all energy in the I (in-phase) arm. If it is assumed that the code replica is perfectly aligned, the multiplication in the I-arm yields the following sum

…Eq 5.14

Where ‘ϕ’ is, phase difference between the phase of the input signal and the phase of the local replica of the carrier. The multiplication in the quadrature arm gives the following:

….Eq 5.15

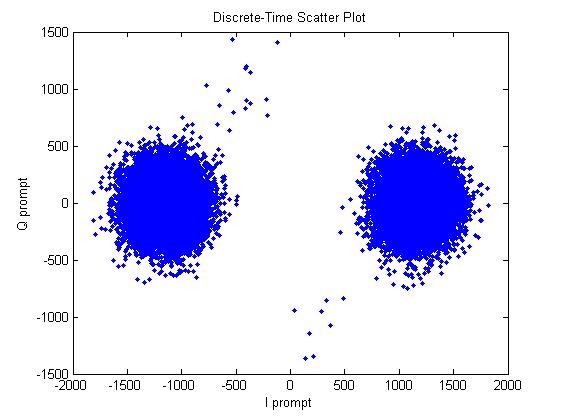
If the two signals are lowpass filtered after the multiplication, the two terms with the double intermediate frequency are eliminated and the following two signals remain:

;

To find a term to feed back to the carrier phase oscillator, the phase error of the local carrier phase replica can be found as

= = tan () where =

From the above equation, it can be seen that the phase error is minimized when the correlation in the quadrature-phase arm is zero and the correlation value in the inphase arm is maximum. The arctan discriminator in above equation is used as the costas loop discriminator.



##### Figure 5.8: Discrete time scatter plot

The behavior of the Costas loop when a 180° phase shift occurs is more clearly illustrated in Figure 5.8. In this figure the vector sum of Ik (I prompt) and Qk (Q prompt) is shown as the vector in the coordinate system. If the local carrier wave is in phase with the input signal, the vector will be aligned to the I- axis. When the signal is tracked correctly, the vector sum of Ik and Qk tends to remain aligned with the I- axis. This property ensures that if a navigation bit transition occurs, the vector on the phasor diagram will flip 180° and the Costas loop will track the signal. This property makes the Costas loop the commonly chosen phase locked loop in GNSS receivers. The output of the phase discriminator is filtered to predict and estimate any relative motion of the satellite and the Doppler frequency.

The problem with using an ordinary PLL is that it is sensitive to 180° phase shifts, due to navigation bit transitions. A PLL used in a GPS or GLONASS receiver has to be insensitive to 180° phase shifts. As the costas loop is insensitive to 180° phase shift is used as a tracking loop for GLONASS receivers.

## 5.4 Code Tracking Loop

The goal of a code tracking loop is to keep track of the code phase of a specific code in the incoming signal. To obtain a perfectly aligned replica of the code, code tracking loop is implemented. This loop consists of early, prompt and late signals from the code generators, filters and discriminators. The incoming IF signal is first down-converted by the carrier frequency to baseband data in I and Q channels. The down- conversion process can be described in Eq. 5.16 and Eq. 5.17

…………………………………………….Eq 5.16

Q …………………………………………….Eq 5.17

Where Fc is the IF carrier frequency estimated by the acquisition program. After multiplying the incoming signal with a perfectly aligned local replica of the carrier wave, the signal is multiplied with three code replicas, namely early, prompt and late, with a spacing of chip. After the second multiplication, the three outputs are integrated and dumped. The output of these integrations gives a numerical value indicating how much the specific code replica correlates with the incoming signal. The correlation values are fed into code discriminator which gives early minus late value to the code loop filter which determines phase change in local prompt code which is given as input to code NCO. The locally generated prompt signal is used to de-spread code in the incoming signal. The early and late codes each correlate with the incoming C/A coded signal to produce two other outputs. These outputs are filtered, squared and compared using a discriminator [2][6].

The code tracking loop in the GLONASSS receiver is a delay lock loop (DLL) called an early–late tracking loop is shown below Figure5.9.

However the DLL with three correlators is optimal when the local carrier wave is locked in phase and frequency, but when there is a phase error on the local carrier wave, the signal will be more noisy, making it more difficult for the DLL to keep lock on the code. So instead of three correlators DLL, six correlators DLL is used in GLONASS software receiver.

I

Q

Incoming signal

NCO

90°

E

P

L

L

P

E

Integrate &Dump

Integrate &Dump

Integrate &Dump

Integrate &Dump

Integrate &Dump

Integrate &Dump

QE

QP

QL

D =

IL

IP

IE

PRN code generator

Loop Filter

##### Figure 5.9: Code tracking loop with six correlators

In this code tracking loop two types of discriminators are used:

1. Coherent discriminator: where all the power is on I (in-phase arm) channel, implying phase lock is achieved.

2. Non-coherent discriminator: where phase lock is not required.

Some types of DLL discriminators are presented in Table 5.1. In this table E and L represent the following equations [6]:

The typical code tracking loop utilizes a normalized early-minus-late-discriminator. In this case, the difference between the power of early and late is calculated and sent to the discriminator. The result is filtered and sent to the code NCO for generating the local PRN code. This difference indicates which one (early or late ranging code) contains more energy and thus whether the NCO must advance or delay the locally generated code is determined. When the power of the early and late correlators is the same, the difference is zero and this is the objective of code tracking loop. Therefore the implemented code discriminator in code tracking loop is the normalized early minus late power. This discriminator is described as

D = ………………………..Eq 5.18

In above equation 5.18, IE and QE, IL and QL are the output from four of the six correlators as shown in Figure5.9. The normalized early minus late power discriminator is chosen because it is independent of the performance of the PLL as it uses both the inphase and quadrature arms. The discriminator can be used with signals with different signal-to-noise ratios and different signal strengths.

###### Table 5.1: Different types of code tracking loop discriminators

|  |  |  |
| --- | --- | --- |
| Type | Discriminator | Characteristics |
| Coherent |  | Simplest of all discriminators.  Does not require the Qbranch but requires a good carrier tracking loop for optimal Functionality. |
| Non Coherent |  | Early minus late power. The discriminator response is nearly the same as the coherent discriminator inside ±1/2 chip. |
|  | Normalized early minus late Power. When the chip error is larger than a ½ chip; this will help the DLL to keep track in noisy signals. |
| ) | Dot product. This is the only DLL discriminator that uses all six correlators outputs. |

By considering the local C/A code, incoming signals auto-correlation has roughly a triangular shape with a typical two-chip space length, the early-minus-late correlators with and without tracking error is shown in Figure 5.10.

From the Figure 5.10, Code tracking loop generates three local code replicas. These three local codes are generated and correlated with the incoming signal. In this case, the chip space between the early and prompt replicas is half a chip. It gives high value only when the prompt code is well aligned, else it results in code phase error which has to be minimized by a code tracking loop.

E

P

L

Incoming signal

00

0.5

-1

1

-0.5

Early

Prompt

Late

Correlation

Chips

Chips

00

0.5

-1

1

-0.5

Correlation

E

L

P

Early

Prompt

Late

Incoming signal

##### Figure 5.10: Correlation of E, P, and L with incoming signal

## 5.5 Combined Code Tracking and Carrier tracking loop

In the previous sections, the code tracking loop and the carrier tracking loop are described in detail [6]. The following figure shows how the code tracking loop and the carrier tracking loop can be combined to minimize the computational load on the receiver. Figure 5.11 shows an optimized version of the combined tracking loops.

Phase discriminator

Loop filter (B, C1, C2)

NCO Carrier generator

90°phase shift

Incoming

Signal

L

P

E

I

Integrate &Dump

Integrate &Dump

Integrate &Dump

Integrate &Dump

Integrate &Dump

Integrate &Dump

D =

PRN code generator

L

P

E

IE

IP

IL

QL

QP

QE

Loop Filter

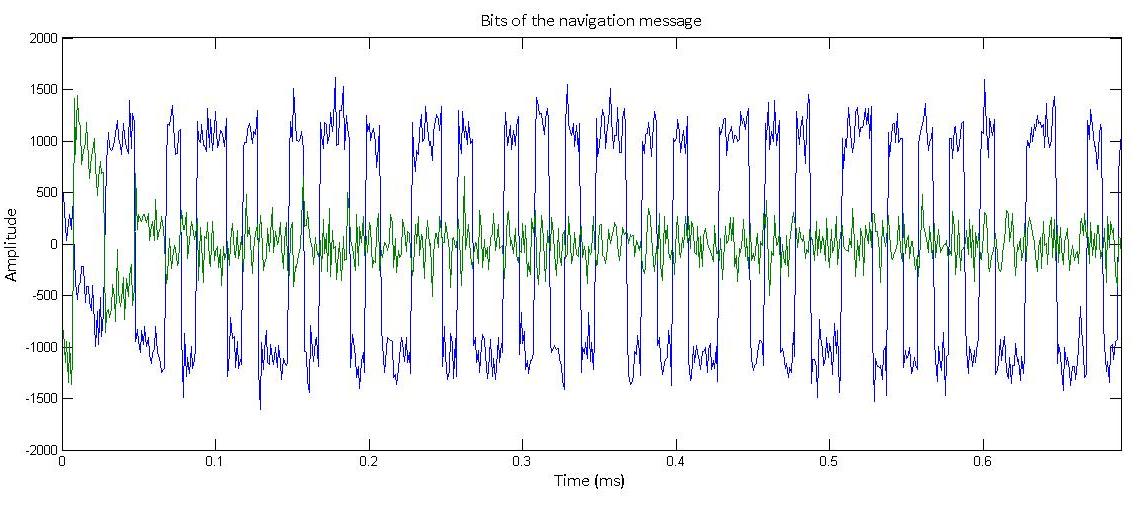
Q

##### Figure 5.11: Simplified complete tracking channel of GLONASS Receiver

Here the inputs to the carrier phase discriminator are the Ip and Qp correlation values from the code tracking loop. The IE and QE, IL and QL are given as inputs to the code phase discriminator (Normalized early minus late discriminator). In this way the three multiplications in the Costas loop are eliminated, and also the computation time is reduced. The complete tracking channel receives the approximate Doppler frequency and code offset for each GLONASS satellite being acquired from acquisition. The tracking channel uses these parameters to generate some measurements which can be used to generate navigation solution. The resulting navigation data solution is needed to extract navigation message to calculate position of satellite and user which will be explained in next sections.

## 5.6 Tracking Results

In Figure 5.12, the demodulated navigation data result from the tracking of one of the four channels is plotted. In this figure, this tracking is showed for 34 seconds (34,000 ms) of data but here some part of navigation data is shown. In the plot, the navigation bits are clearly visible. The transitions of the navigation bits are sharp and it is easy to decide if the bits are positive or negative. The conversion between these values and the navigation bits are determined after finding the time marks in navigation data decoding. In the plot green color shows Q channel noise in tracking channel.



##### Figure 5.12: Tracking results navigation data bits of Satellite PRN 3

## 5.7 Conclusion

This chapter describes about basic demodulation of satellite signal and navigation data decoding. The demodulation of data can be done through tracking. It can be implemented by using a second order DLL and PLL loops, which have been discussed in detail and implemented. These loops in tracking are used to generate exact local code and local carrier inside the receiver, for the demodulation of data bits in the incoming signal. After tracking the resulting prompt values or navigation data bits are used in navigation data extraction to decode the navigation solutions.

# 6. Time synchronous Implementation

## 6.1 Introduction

To develop GLONASS software receiver in C need to follow the time synchronous implementation in tracking and navigation data decoding.

## 6.2 Time synchronous Implementation

The main goal of time synchronous is, while tracking every millisecond of data signal it has to calculate the pseudoranges. It occurs only when the correct time mark or string start is identified otherwise it has to track every millisecond to get time mark or string start.

In time synchronous implementation there are three stages first stage is acquisition. In the acquisition process need to estimate the codephase offset and carrier frequency Doppler of GLONASS satellite signals. In this process the GLONASS software receiver code in MATLAB is utilized, but it is converted in to C by using same algorithm without any modification. For each satellite in the acquisition list, the codephase is transformed in to the frequency domain. Here carrier frequency has been removed through Doppler frequency search of the nominal frequency of the channel. The resulting signal is converted into the frequency domain. The result is multiplied by the frequency domain version of the code. The inverse DFT of the last result is the correlation of the code and the baseband signal.

The remaining code is to find the maximum value in the acquisition search space. If the maximum of the correlation and the second peak ratio is bigger than the threshold defined in the settings structure, the signal is detected in the data and the result of the acquisition are refined using a longer data in fine frequency resolution. Therefore from the acquisition, estimation of codephase, carrier frequency can be obtained. After the acquisition process completed, the estimated parameters are given to input to the second stage i.e. the tracking module.

In tracking module MATLAB tracking code is converted into C. To make into time synchronous the tracking module has modified so that the tracking module contains four modules those are:

**(1) Feedback parameters**

Acquisition estimated parameters are the feedback parameters to the tracking module. These parameters change for every tracking iteration and need to be updated continuously to lock the receiver with an incoming signal. These parameters are calculated by the function initfeedbackTrack () in time synchronous code.

**(2) Fixed parameters**

Fixed parameters are constant parameters which includes Filter coefficients, PLL variables, DLL variables or parameters. The filter coefficients can be calculated by using calculation of loop coefficients function. In time synchronous these parameters can be calculated by the function initFixedParameters ().

**(i) Calculation of Loop coefficients:**

The tracking loops use second order control system and their parameters such as Noise bandwidth, natural frequency are calculated using the function calcLoopCoeff ().

**(3) Tracking Iterations**

The TrackIter module consists of carrier tacking loop, code tracking loop implementation, to generate local carrier and local code respectively.

**(i) Carrier Tracking**

The track iteration is composed of carrier and code tracking. A PLL is used to track the carrier which can be implemented by costas loop. The loop calculates the error in phases of the received signal and locally generated signal and tends to minimize it. A Costas loop is used as it is insensitive to the 180 degree phase changes of the data. Once the loop is locked, the signal energy lies within the I-component of the signal.

**(ii) Code Tracking**

A DLL is usually used to track variations in the codephase. The loop calculates the codephase error obtained due to the misalignment of the locally generated code with the code of the received signal and keeps the error in prescribed limits. This is achieved by correlating the received signal with three time delayed versions called Early, Late and Prompt codes which are advanced, delayed by 0.5 chips respectively. The three versions of the code are multiplied with the I and Q components that are generated locally.

The error for both the loops is estimated using the expression

Carr phase Error = and Code phase Error =

The feedback variables are updated for every iteration based on the error signals obtained using the above formula.

**(4) Track Results**

After performing track iterations, the navigation bits are located in inphase arm of the signal. Similarly PLL discriminator results, DLL discriminator results are stored, but here navigation data bits are needed to find the position of receiver on the earth.

## 6.3 Time synchronous Pseudo code for Acquisition and Tracking in C

### 6.3.1 Pseudo Code for Acquisition

#include<stdio.h>

#include<conio.h>

#include"settings.h"

#include"acqresults.h"

#include"generateCAcode.h"

#include <stdlib.h>

#include "fft.h"

#include<math.h>

#define PI 3.141592653589793

unsigned int samplesPerCode;

int FFTLength;

Settings s1;

acqresults prnresults[15];

void makeCAtable(short \*p2caCode, COMPLEX \* p2LocalcaCode);

void main()

{

int i,FFTLength,j,a,k,samplesPerCodeChip,excludeRangeIndex1,excludeRangeIndex2;

FILE \*fptest,\*fptest2,\*fp,\*fptest3,\*fptest4,\*fptest5,\*fptest6;

short CAcode[511],\*codeValueIndex,\*longCaCode;

COMPLEX \*p2temp,\*p2LocalcaCode,\*p2caCodeFFT,\*p2caCode,\*longsignal,\*signal0DC,\*xCarrier,\*xCarrier\_tmp,\*output;

int \*p2frqBins,\*b;

double \*\*p2results;

int numberOfFrqBins,frequencyBinIndex,codePhase,\*p2codePhaseRange;

double p2sincarrier,p2coscarrier,\*p2acqRes1,\*p2acqRes2,max1,max2,p2maxres1[141],peaksize,\*p2maxres2,uniqFftPts,fftMax,fftMaxIndex,max,\*fftFreqBinsRev;

char PRN = 0,dataAdaptCoeff;

double phasePoints[80000],ts,fftNumPts,size,\*fftxc,\*fftFreqBins,x1,x2,mean1,mean2;

COMPLEX \*p2signal1, \*p2signal2,\*I1,\*Q1,\*I2,\*Q2,\*Z1,\*Z2,\*p2IQfreqDom1,\*p2IQfreqDom2,\*p2convCodeIQ1,\*p2convCodeIQ2,\*p2caCodeFreqDom,\*p2ifft1,\*p2ifft2;

char \*data,\*p2data,\*p2longsignal;

int temp1,temp2,temp3;

printf("Acquiring Satellites\n");

//opening the GLONASS signal file

fp=fopen("C:\\Users\\home\\Desktop\\PROJECT\\GlonassSecondFiles\\Acquisition\\FFF005.DAT","rb");

//Initialising the variables of the structure Settings

initSettings();

//To generate the C/A code

generateCAcode(CAcode);

samplesPerCode = floor(s1.samplingFreq /(s1.codeFreqBasis /s1.codeLength));

FFTLength=pow(2,ceil(logf(5\*samplesPerCode)/logf(2)));

p2LocalcaCode = (COMPLEX \*) calloc(FFTLength, sizeof(COMPLEX));

p2caCodeFFT = (COMPLEX \*) calloc(FFTLength, sizeof(COMPLEX));

p2caCode= (COMPLEX \*) calloc(samplesPerCode, sizeof(COMPLEX));

if (s1.fileType==1)

dataAdaptCoeff=1;

else

dataAdaptCoeff=2;

//to skip the required number of bytes in the signal to be read

fseek(fp,dataAdaptCoeff\*s1.skipNumberOfBytes,1);

data = (char \*) calloc(11\*samplesPerCode\*2\*dataAdaptCoeff,sizeof(char));

p2signal1 = (COMPLEX \*) calloc(5\*samplesPerCode, sizeof(COMPLEX));

p2signal2 = (COMPLEX \*) calloc(5\*samplesPerCode, sizeof(COMPLEX));

p2caCodeFreqDom = (COMPLEX \*)calloc(FFTLength,sizeof(COMPLEX));

longsignal = (COMPLEX \*) calloc(11\*samplesPerCode, sizeof(COMPLEX));

signal0DC = (COMPLEX \*) calloc(11\*samplesPerCode, sizeof(COMPLEX));

numberOfFrqBins = (s1.acqSearchBand \* 10) + 1;

ts = 1/s1.samplingFreq;

//reading the data from the signal file

if (dataAdaptCoeff==2)

{

a=fread(data,1,11\*dataAdaptCoeff\*samplesPerCode,fp);

}

p2data = data;

p2longsignal = data;

for(j=0;j<5\*samplesPerCode;j++)

{

p2signal1[j].real = (double) \*p2data++;

p2signal1[j].Imag = -(double) \*p2data++;

}

for(j=0;j<5\*samplesPerCode;j++)

{

p2signal2[j].real = (double) \*p2data++;

p2signal2[j].Imag = -(double) \*p2data++;

}

for(j=0;j<11\*samplesPerCode;j++)

{

longsignal[j].real = (double) \*p2longsignal++;

longsignal[j].Imag = (double) \*p2longsignal++;

}

x1 = x2 = 0;

for(j=0;j<11\*samplesPerCode;j++)

{

x1 = longsignal[j].real + x1;

x2 = longsignal[j].Imag + x2;

}

mean1 = x1/(11\*samplesPerCode);

mean2 = x2/(11\*samplesPerCode);

for(j=0;j<11\*samplesPerCode;j++)

{

signal0DC[j].real = longsignal[j].real - mean1;

signal0DC[j].Imag = longsignal[j].Imag - mean2;

}

//to generate the CAtable for the C/A code

makeCAtable(CAcode,p2caCode);

//to generate the CAtable

for(i=0,j=0;(j<=samplesPerCode) && (i<=samplesPerCode\*5);j++,i++)

{

if(j==samplesPerCode)

j = 0;

p2LocalcaCode[i] = p2caCode[j];

}

for(i=5\*samplesPerCode;i<FFTLength;i++)

{

p2LocalcaCode[i].real = 0;

p2LocalcaCode[i].Imag = 0;

}

fft\_ifft(p2LocalcaCode,p2caCodeFFT,FFTLength,1);

conjugate(p2caCodeFFT,p2caCodeFreqDom);

for(PRN = 1 ; PRN <= NO\_OF\_SATLLITES; PRN++)

{

p2frqBins = (int \*) calloc(numberOfFrqBins,sizeof(int));

I1 = (COMPLEX \*)calloc(5\*samplesPerCode,sizeof(COMPLEX));

Q1 = (COMPLEX \*)calloc(5\*samplesPerCode,sizeof(COMPLEX));

I2 = (COMPLEX \*)calloc(5\*samplesPerCode,sizeof(COMPLEX));

Q2 = (COMPLEX \*)calloc(5\*samplesPerCode,sizeof(COMPLEX));

Z1 = (COMPLEX \*)calloc(FFTLength,sizeof(COMPLEX));

Z2 = (COMPLEX \*)calloc(FFTLength,sizeof(COMPLEX));

p2IQfreqDom1 = (COMPLEX \*)calloc(FFTLength,sizeof(COMPLEX));

p2IQfreqDom2 = (COMPLEX \*)calloc(FFTLength,sizeof(COMPLEX));

p2convCodeIQ1 = (COMPLEX \*)calloc(FFTLength,sizeof(COMPLEX));

p2convCodeIQ2 = (COMPLEX \*)calloc(FFTLength,sizeof(COMPLEX));

p2ifft1 = (COMPLEX \*)calloc(FFTLength,sizeof(COMPLEX));

p2ifft2 = (COMPLEX \*)calloc(FFTLength,sizeof(COMPLEX));

p2acqRes1 = (double \*)calloc(FFTLength,sizeof(double));

p2acqRes2 = (double \*)calloc(FFTLength,sizeof(double));

p2codePhaseRange = (int \*)calloc((samplesPerCode),sizeof(int));

p2maxres2 = (double \*)calloc(samplesPerCode,sizeof(double));

b = (int \*)calloc(10\*samplesPerCode,sizeof(int));

p2results = (double \*\*)malloc(141 \* sizeof(double \*));

for(i = 0; i < numberOfFrqBins; i++)

{

p2results[i] = (double \*) malloc(samplesPerCode \* sizeof(double));

}

for(k=0;k<numberOfFrqBins;k++)

{

p2frqBins[k] = (s1.IF+PRN\*0.5625e6)-(s1.acqSearchBand/2)\*1000+100\*(k);

for(i=0;i<5\*samplesPerCode;i++)

{

phasePoints[i] = 2\*PI\*i\*ts;

p2sincarrier = sin(p2frqBins[k] \* phasePoints[i]);

p2coscarrier = cos(p2frqBins[k] \* phasePoints[i]);

I1[i].real = p2sincarrier\*p2signal1[i].real;

I1[i].Imag = p2sincarrier\*p2signal1[i].Imag;

Q1[i].real = p2coscarrier\*p2signal1[i].real;

Q1[i].Imag = p2coscarrier\*p2signal1[i].Imag;

I2[i].real = p2sincarrier\*p2signal2[i].real;

I2[i].Imag = p2sincarrier\*p2signal2[i].Imag;

Q2[i].real = p2coscarrier\*p2signal2[i].real;

Q2[i].Imag = p2coscarrier\*p2signal2[i].Imag;

}

c\_addsignals(I1,Q1,Z1);

c\_addsignals(I2,Q2,Z2);

for(i=5\*samplesPerCode;i<FFTLength;i++)

{

Z1[i].real = 0;

Z1[i].Imag = 0;

Z2[i].real = 0;

Z2[i].Imag = 0;

}

fft\_ifft(Z1,p2IQfreqDom1,FFTLength,1);

fft\_ifft(Z2,p2IQfreqDom2,FFTLength,1);

c\_multsignals(p2IQfreqDom1,p2caCodeFreqDom,p2convCodeIQ1);

c\_multsignals(p2IQfreqDom2,p2caCodeFreqDom,p2convCodeIQ2);

fft\_ifft(p2convCodeIQ1, p2ifft1,FFTLength,-1);

fft\_ifft(p2convCodeIQ2, p2ifft2,FFTLength,-1);

c\_absolute(p2ifft1,p2acqRes1);

c\_absolute(p2ifft2,p2acqRes2);

max1 = p2acqRes1[0];

max2 = p2acqRes2[0];

for(i=0;i<samplesPerCode;i++)

{

if(max1 < p2acqRes1[i])

max1 = p2acqRes1[i];

if(max2 < p2acqRes2[i])

max2 = p2acqRes2[i];

}

if(max1 > max2)

{

for(j=0;j<samplesPerCode;j++)

p2results[k][j] = p2acqRes1[j];

}

else

for(j=0;j<samplesPerCode;j++)

p2results[k][j] = p2acqRes2[j];

}

for(i=0,peaksize=0;i<(numberOfFrqBins);i++)

{

for(j=samplesPerCode,p2maxres1[i] = 0;j>0;j--)

{

if(p2results[i][j-1] > p2maxres1[i])

p2maxres1[i] = p2results[i][j-1];

}

if( ( peaksize < p2maxres1[i]))

{

peaksize = p2maxres1[i];

frequencyBinIndex = i+1;

}

}

for(j=0,peaksize=0;j<(samplesPerCode);j++)

{

for(i=numberOfFrqBins,p2maxres2[i] = 0;i>0;i--)

{

if(p2results[i-1][j] > p2maxres2[j])

p2maxres2[j] = p2results[i-1][j];

}

if( ( peaksize < p2maxres2[j]))

{

peaksize = p2maxres2[j];

codePhase = j+1;

}

}

samplesPerCodeChip = floor(s1.samplingFreq/s1.codeFreqBasis);

excludeRangeIndex1 = codePhase - samplesPerCodeChip;

excludeRangeIndex2 = codePhase + samplesPerCodeChip;

if (excludeRangeIndex1 < 2)

{

for(i = excludeRangeIndex2;i<(samplesPerCode + excludeRangeIndex1);i++)

p2codePhaseRange[i] = i;

}

else if (excludeRangeIndex2 >= samplesPerCode)

{

for(i=(excludeRangeIndex2 - samplesPerCode);i<=excludeRangeIndex1;i++)

p2codePhaseRange[i] = i;

}

else

{

for(i=1,j=1;(i<=samplesPerCode) ;i++)

{

if((i<=excludeRangeIndex1)||(i>=excludeRangeIndex2))

{

p2codePhaseRange[j] = i;

j++;

}

}

temp2 = j - 1;

}

for(i=1,j=1,temp1 = p2results[frequencyBinIndex-1][j];i<=temp2;i++,j=p2codePhaseRange[i])

{

if((i>1) && (temp1 < p2results[frequencyBinIndex-1][j]))

{

temp1 = p2results[frequencyBinIndex-1][j];

temp3 = j;

}

}

prnresults[PRN].peakMetric = peaksize/temp1;

if ( prnresults[PRN].peakMetric > s1.acqThreshold)

{

printf("%d\n",PRN);

generateCAcode(CAcode);

xCarrier = (COMPLEX \*) calloc(10\*samplesPerCode, sizeof(COMPLEX));

codeValueIndex = (short \*)calloc(10\*samplesPerCode,sizeof(short));

longCaCode = (short \*)calloc(10\*samplesPerCode,sizeof(short));

for(i=1;i<=10\*samplesPerCode;i++)

{

codeValueIndex[i-1] = floor((ts \* (i))/(1/s1.codeFreqBasis));

b[i-1] = (int)floor(codeValueIndex[i-1]/511);

longCaCode[i-1] = CAcode[(codeValueIndex[i-1]-(b[i-1])\*511)];

}

for(i=codePhase,j=0;i<=(codePhase + (10\*samplesPerCode)-1)&&j<10\*samplesPerCode;i++,j++)

{

xCarrier[j].real = signal0DC[i].real\*(double)longCaCode[j];

xCarrier[j].Imag = signal0DC[i].Imag\*(double)longCaCode[j];

}

size=0;

for(i=0;i<10\*samplesPerCode;i++)

size = (sizeof(xCarrier[i]))+size;

size = size/16;

fftNumPts = 8\*pow(2,ceil((logf(size)/(logf(2)))));

xCarrier\_tmp = (COMPLEX \*) calloc(fftNumPts,sizeof(COMPLEX));

output = (COMPLEX \*) calloc(fftNumPts,sizeof(COMPLEX));

fftxc = (double \*)calloc(fftNumPts,sizeof(double));

for(i=0;i<size;i++)

{

xCarrier\_tmp[i].real = xCarrier[i].real;

xCarrier\_tmp[i].Imag = xCarrier[i].Imag;

}

fft\_ifft(xCarrier\_tmp,output,fftNumPts,1);

for(i=0;i<fftNumPts;i++)

{

fftxc[i] = sqrt(output[i].real\*output[i].real+output[i].Imag\*output[i].Imag);

}

uniqFftPts = ceil((fftNumPts + 1) / 2);

fftFreqBins = (double \*)calloc((uniqFftPts-1),sizeof(double));

fftFreqBinsRev = (double \*)calloc((uniqFftPts-1),sizeof(double));

fftMax = fftxc[0];

for(i=1;i<fftNumPts;i++)

{

if(fftMax < fftxc[i])

{

fftMax = fftxc[i];

fftMaxIndex = i;

}

if(fftMax == fftxc[0])

fftMaxIndex=0;

}

for(i=0;i<=(uniqFftPts-1);i++)

{

fftFreqBins[i] = i \* s1.samplingFreq/fftNumPts;

}

if (fftMaxIndex > uniqFftPts)

{

if ( (fftNumPts-floor(fftNumPts/2)\*2) == 0 )

{

for(i=uniqFftPts-1,j=0;i>=1;i--,j++)

{

fftFreqBinsRev[j] = -fftFreqBins[i];

}

fftMax = fftxc[(int)uniqFftPts];

for(i=(uniqFftPts+1),j=1;i<fftNumPts;i++,j++)

{

if(fftMax < fftxc[i])

{

fftMax = fftxc[i];

fftMaxIndex = j;

}

if(fftMax == fftxc[(int)uniqFftPts])

fftMaxIndex = 0;

}

prnresults[PRN].carrfreq = -fftFreqBinsRev[(int)fftMaxIndex];

}

else

{

for(i=uniqFftPts;i>1;i--)

{

fftFreqBinsRev[i] = -fftFreqBins[i];

}

fftMax = fftxc[(int)uniqFftPts];

for(i=(uniqFftPts+1),j=0;i<fftNumPts;i++,j++)

{

if(fftMax < fftxc[i])

{

fftMax = fftxc[i];

fftMaxIndex = j;

}

if(fftMax == fftxc[(int)uniqFftPts])

fftMaxIndex = 0;

}

prnresults[PRN].carrfreq = p2frqBins[frequencyBinIndex-1];

}

}

else

prnresults[PRN].carrfreq = pow(-1,(s1.fileType-1))\*fftFreqBins[(int)fftMaxIndex];

prnresults[PRN].codePhase = codePhase;

free(xCarrier);

free(codeValueIndex);

free(longCaCode);

free(xCarrier\_tmp);

free(output);

free(fftxc);

}

else

{

prnresults[PRN].carrfreq = p2frqBins[frequencyBinIndex-1];

prnresults[PRN].codePhase = codePhase;

printf(".");

}

free(p2frqBins);

free(I1);

free(Q1);

free(I2);

free(Q2);

free(Z1);

free(Z2);

free(p2IQfreqDom1);

free(p2IQfreqDom2);

free(p2convCodeIQ1);

free(p2convCodeIQ2);

free(p2ifft1);

free(p2ifft2);

free(p2acqRes1);

free(p2acqRes2);

free(p2codePhaseRange);

free(p2maxres2);

free(p2results);

}

}

### 6.3.2 Pseudo Code for Tracking

#include<stdio.h>

#include"generateCACode.h"

#include"initTrackResults.h"

#include<stdlib.h>

Settings s1;

short channelNr,\*\*CAcode,g=0;

long z=0;

void main()

{

int subFrameStart[7],j,i,s=0,loopCnt;

double codePeriods;

char dataAdaptCoeff,channelNr,a,svnum;

short \*Cacode;

FILE \*fid, \*fptest, \*fptest1, \*fptest2, \*fptest3, \*fptest4, \*fptest5, \*fptest6, \*fptest7,\*fptest8,\*fptest9,\*fptest10,\*fptest11,\*fptest12;

struct Feedback\_Parameters \*FEEDBACK\_PARAMETERS,FeedBackParameters[7];

struct FixedParameters \*FixedParameters;

struct trackResults \*trackResults;

Cacode = (short \*)calloc(513,sizeof(short));

CAcode = (short \*\*)malloc(14 \* sizeof(short \*));

for(i = 0; i < 14; i++)

{

CAcode[i] = (short \*) malloc(511 \* sizeof(short));

}

printf("Starting processing...");

fid = fopen ("C:\\Users\\home\\Desktop\\PROJECT\\GlonassSecondFiles\\Acquisition\\FFF005.DAT","rb");

//Initialize the multiplier to adjust for the data type

if (s1.fileType==1)

dataAdaptCoeff=1;

else

dataAdaptCoeff=2;

AcqResults();

initSettings();

Cacode = generateCAcode(Cacode);

for(i=0;i<14;i++)

{

for(j=0;j<511;j++)

{

CAcode[i][j] = Cacode[j];

}

}

// Initialize channels and prepare for the run

// Start further processing only if a GNSS signal was acquired (the

// field FREQUENCY will be set to 0 for all not acquired signals)

for(i=0;i<14;i++)

{

if (acqresults[i].carrFreq != 0)

{

a = 1;

}

}

if(a==1)

{

preRun(acqresults,s1);

showChannelStatus(Channel, s1);

}

else

{

// No satellites to track, exit

printf("No GNSS signals detected, signal processing finished.\n");

}

FEEDBACK\_PARAMETERS = initFeddBackTrack(s1,Channel);

trackResults = initTrackResults(s1);

FixedParameters = initFixedParameters(s1);

loopCnt = 0;

for(i=0;i<s1.msToProcess;i++)

{

loopCnt = loopCnt+1;

if (loopCnt%100==0)

printf("%d\n",loopCnt);

for (channelNr = 0;channelNr<s1.numberOfChannels;channelNr++)

{

if (Channel[channelNr].PRN != 0)

{

trackResults[channelNr].PRN = Channel[channelNr].PRN;

FEEDBACK\_PARAMETERS[channelNr].PRN = Channel[channelNr].PRN;

FeedBackParameters[channelNr] = FEEDBACK\_PARAMETERS[channelNr];

FeedBackParameters[channelNr] = TrackIter (fid, FeedBackParameters[channelNr], FixedParameters[channelNr], s1,Channel);

FEEDBACK\_PARAMETERS[channelNr] = FeedBackParameters[channelNr];

trackResults[channelNr].I\_E[loopCnt] = FeedBackParameters [channelNr].I\_E;

trackResults[channelNr].I\_L[loopCnt] = FeedBackParameters [channelNr].I\_L;

trackResults[channelNr].I\_P[loopCnt] = FeedBackParameters [channelNr].I\_P;

trackResults[channelNr].Q\_E[loopCnt] = FeedBackParameters [channelNr].Q\_E;

trackResults[channelNr].Q\_L[loopCnt] = FeedBackParameters [channelNr].Q\_L;

trackResults[channelNr].Q\_P[loopCnt] = FeedBackParameters [channelNr].Q\_P;

trackResults[channelNr].pllDiscr[loopCnt] = FeedBackParameters [channelNr].pllDiscr;

trackResults[channelNr].pllDiscrFilt[loopCnt] = FeedBackParameters[channelNr].pllDiscrFilt;

trackResults[channelNr].dllDiscr[loopCnt] = FeedBackParameters [channelNr].dllDiscr;

trackResults[channelNr].dllDiscrFilt[loopCnt] = FeedBackParameters [channelNr].dllDiscrFilt;

trackResults[channelNr].absoluteSample[loopCnt] = FeedBackParameters[channelNr].absoluteSample/dataAdaptCoeff;

trackResults[channelNr].status = Channel[channelNr].status;

}

}

}

fclose(fptest);

fclose(fid);

}

## 6.4 Description of C functions and structures used in Acquisition and Tracking

### 6.4.1 Structures :

**Settings :**

It consists of msToProcess, numberOfChannels, skipNumberOfBytes, fileType, IF, samplingFreq, codeFreqBasis, codeLength, skipAcquisition, acqSearchBand, acqThreshold fields which are assigned to the predetermined values in the function initSettings( ).

**COMPLEX :**

It consists of real and imaginary fields and the absolute value, exponential, addition, subtraction, multiplication, conjugate of the variables of this structure type are performed using double c\_abs(COMPLEX \*x), double c\_power(COMPLEX \*x), double c\_angle(COMPLEX \*x); void c\_add(COMPLEX \*x,COMPLEX \* y,COMPLEX \* z), void c\_sub(COMPLEX \*x, COMPLEX \* y, COMPLEX \* z), void c\_addsignals(COMPLEX \*x, COMPLEX \* y, COMPLEX \*z), void c\_mult(COMPLEX \*x, COMPLEX \* y,COMPLEX \* z), void c\_multsignals(COMPLEX \*x, COMPLEX \*y, COMPLEX \*z), void conjugate(COMPLEX \*x,COMPLEX \*y), void c\_absolute(COMPLEX \*x,double \*y) functions.

**acqResults :**

It consists of carrFreq, codePhase, peakMetric fields which are assigned to the carrier frequency, codephase, peakmetric of the visible satellites obtained from acquisition using AcqResults( ) function.

**Channel :**

It consists of PRN, acquiredFreq, codePhase, status fields and the initialization of tracking channels from acquisition data is performed using void preRun(struct acqResults acqresults[ ],Settings s1) function. The acquired signals are sorted according to the signal strength. This function can be modified to use other satellite selection algorithms or to introduce acquired signal properties offsets for testing purposes.

**Feedback\_Parameters :**

It consists of codeFreq, remCodePhase, remCarrPhase, oldCodeNco, oldCodeError, oldCarrNco, oldCarrError, I\_P, I\_E, I\_L, Q\_P, Q\_E, Q\_L, pllDiscrFilt, pllDiscr, dllDiscrFilt, dllDiscr, carrFreq, carrFreqBasis, absoluteSample, PRN fields which are initialized to the fields of the channels allocated for the visible satellites using struct Feedback\_Parameters \* initFeddBackTrack(Settings s1,struct channel Ch[ ]) function.

**FixedParameters :**

It consists of earlyLateSpc, PDIcode, tau1code, tau2code, PDIcarr, tau1carr, tau2carr fields which are assigned the calculated values in struct FixedParameters \*initFixedParameters(Settings s1) function.

**trackResults :**

It consists of status, absoluteSample[MsToProcess], codeFreq[MsToProcess], carrFreq[MsToProcess], I\_P[MsToProcess], I\_E[MsToProcess], I\_L[MsToProcess], Q\_E[MsToProcess], Q\_P[MsToProcess], Q\_L[MsToProcess], dllDiscr[MsToProcess], dllDiscrFilt[MsToProcess], pllDiscr[MsToProcess], pllDiscrFilt[MsToProcess], PRN fields which are initialized to zeros using struct trackResults \*initTrackResults(Settings s1) function which are then modified as per the feedback parameters and the allocated channel parameters variation.

### 6.4.2 Functions :

**void generateCAcode(short \*)** **:**

It is used to generate C/A code of 511bits long.

**void makeCAtable(short \*, COMPLEX \*) :**

It is used to generate all C/A codes and sample them according to the sampling frequency.

**void fft\_ifft(COMPLEX \*p2x, COMPLEX \*p2Y, int N, int select) :**

It is used to convert the baseband signal into frequency domain signal where select variable in the argument specifies the operation i.e., to perform FFT or inverse FFTof the time domain signal.

**void showChannelStatus(struct channel ch[ ], Settings s1) :**

It is used to display the Channel Number, PRN, Frequency, Doppler shift, Code offset, Status of the channels which are allocated for the visible satellites.

**float\* calcLoopCoef(char, float, float) :**

It is used to find the loop coefficients which are used in the PLL’s and DLL’s. Loop noise bandwidth, Damping ratio, Loop gain are the inputs to this function.

**struct Feedback\_Parameters TrackIter(FILE \*, struct Feedback\_Parameters FB, struct FixedParameters FP, Settings s1, struct channel ch[ ]) :**

It is used to implement the carrier and code demodulation using PLL and DLL and the feedback parameters are modified.

## 6.5 Results

Comparision of Acquisition results of visible satellites in MATLAB and C :

Table 6.1: Comparision of Acquisition results in MATLAB and C

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **PRN** | **Carrier Frequency** | | **CodePhase** | | **PeakMetric** | |
| MATLAB  (1.0e+006 \*) | C | MATLAB | C | MATLAB | C |
| 3 | -1.8119 | -1811889.648438 | 6999 | 6998.000000 | 14.7010 | 14.733392 |
| 4 | -1.2454 | -1245384.216309 | 3750 | 3749.000000 | 18.4706 | 18.501371 |
| 5 | -0.6826 | -682647.705078 | 249 | 248.000000 | 7.2975 | 7.304984 |
| 9 | 1.5631 | 1563110.351563 | 2117 | 2116.000000 | 11.7625 | 11.788225 |
| 10 | 2.1236 | 2123588.562012 | 7545 | 7544.000000 | 36.0604 | 36.134478 |
| 11 | 2.6841 | 2684150.695801 | 7300 | 7299.000000 | 10.5320 | 10.562738 |

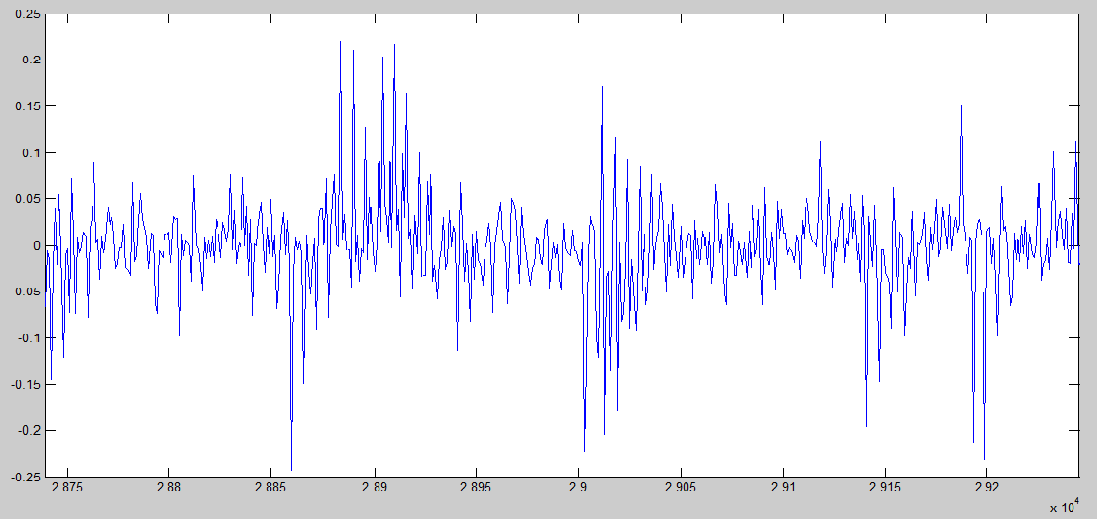
Comparision of Tracking results for 2millisec data in MATLAB and C :

Table 6.2: Comparision of Tracking results for 2msec in MATLAB and C

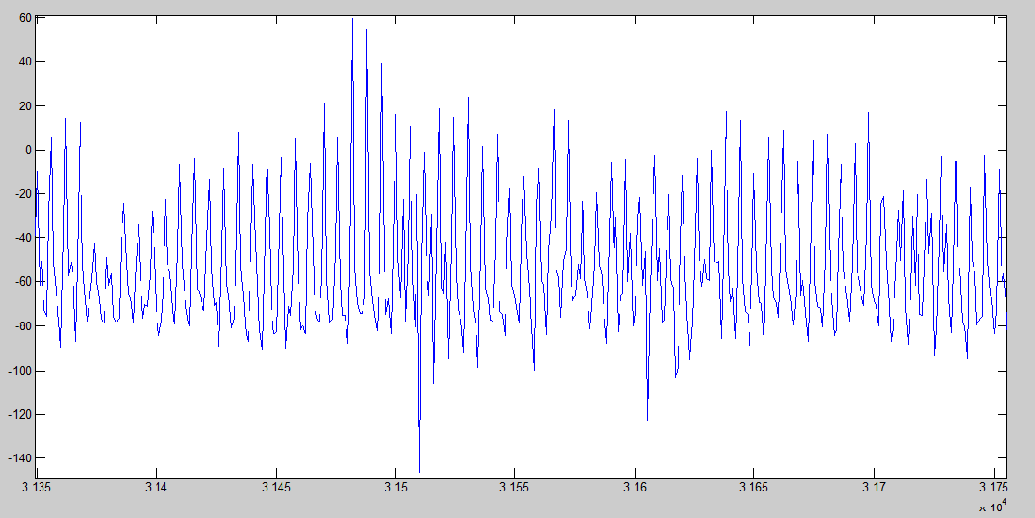
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **I\_P** | | **Absolute Sample** | | **PRN** | |
| MATLAB | C | MATLAB | C | MATLAB | C |
| 1.4642e+003  -1.1541e+003  -107.0158  490.1276  -295.3844  749.0583  1.6667e+003  -939.7027  520.2446  178.1765  -322.4081  -696.2156 | 1471.823987  -1154.096608  -106.926772  467.091361  -265.004199  749.051033  1673.667688  -939.875487  520.554950  101.456716  -269.426914  -696.209967 | 16023544  16019749  16022998  16018116  16023299  16016248  16039545  16035750  16038999  16034116  16039299  16032249 | 16023544  16019749  16022998  16018116  16023299  16016248  16039545  16035750  16038999  16034116  16039299  16032249 | 10  4  3  9  11  5  10  4  3  9  11  5 | 9  3  2  8  10  4  9  3  2  8  10  4 |

**Plots of Tracking Results in C (Time(sec) Vs Amplitude) :**

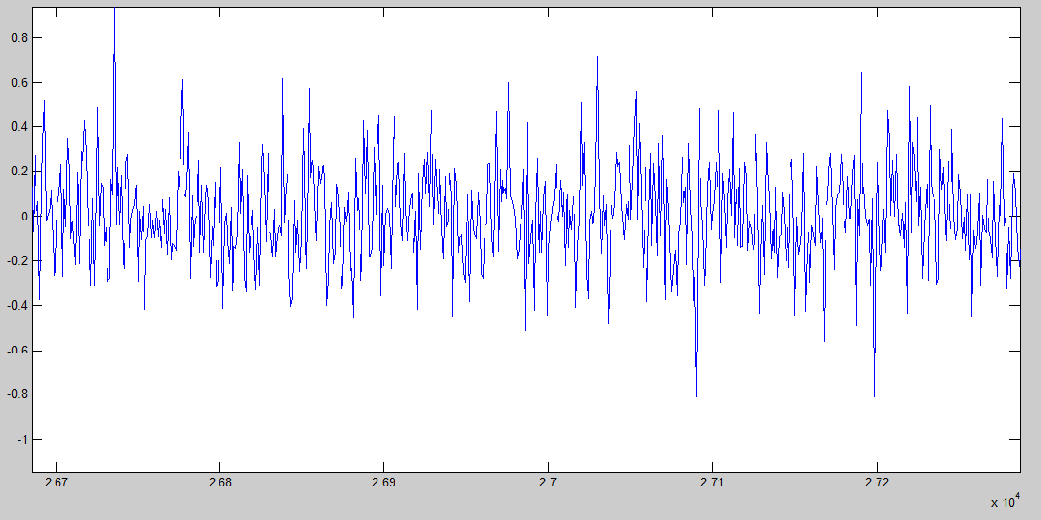
**pllDiscr output**



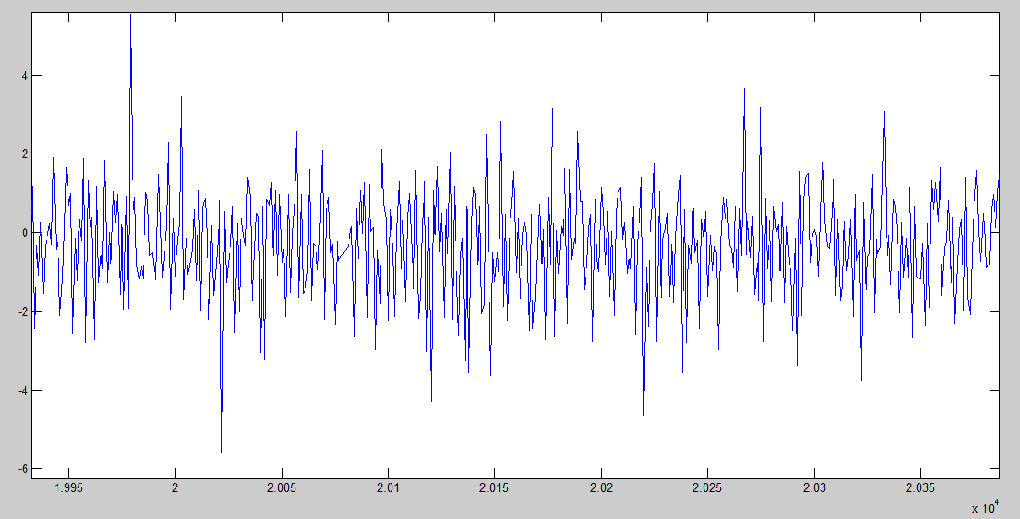
**pllDiscrFilt output**

****

**dllDiscr output**

****

**dllDiscrFilt output**



# 7. Conclusion and Future Scope

Global Navigation Satellite System has a wide research area and can be implemented in various real time applications. In this project focused for the implementation of Acquisition and Tracking algorithms in Microsoft Visual Studio by taking the reference from the MATLAB code. In this development process first the MATLAB code was studied, and it has been converted into C.

The goal of this work is to develop a C code for Acquisition and Tracking of GLONASS Software Receiver. The base band processing of GLONASS software receiver includes acquisition; tracking and additional decoding algorithms to compute receiver position for every millisecond. Here the acquisition is performed by using FFT search method and estimated codephase, carrier frequency values. After acquisition process completed, the estimated parameters are fed into the input of the tracking module. In this module two tracking loops have been introduced and utilized to produce exact carrier and code for the demodulation of navigation data bits in the incoming signal. The results obtained in MATLAB and C-code for Acquisition and Tracking algorithms are in agreement.

**FUTURE SCOPE**

The major challenge in a GLONASS software receiver is “making the acquisition, tracking and navigation data decoding algorithms to run in real time”. The project can be further developed to work on real time data by implementing it in any Floating Point Digital Signal Processor. So the present C code of time synchronous Acquisition and Tracking of GLONASS Software Receiver can be taken as basis for implementing on any Floating Point Digital Signal Processor after developing the code even for Data Decoding.

# References

1. *S. Abbasian Nik and M.G. Petovello* “Multichannel Dual Frequency GLONASS Software Receiver” *Position, Location and Navigation (PLAN) Group, Department of Geomatics Engineering, Schulich School of Engineering, University of Calgary. ION GNSS 2008, Session B4, Savannah, GA, 16-19 September 2008.*
2. *Saloomeh Abbasiannik* *thesis* *on* “Multichannel Dual Frequency GLONASS Software Receiver in Combination with GPS L1 C/A” *by April 2009.*
3. *James Bao-Yen Tsui*, “Fundamentals of Global Positioning System Receivers”: *A Software Approach, 2nd ed. New York: Jhon Wiley & Sons Inc, pp. 108-109, 2000.*
4. *Yuying Zheng School of Electronic and Information Engineering* , “A Software-based Frequency Domain Parallel Acquisition Algorithm for GPS Signal” : *Xi’an Jiaotong University Xi’an 710049,©2010 IEEE, China.*
5. *Filipe Jorge Coelho thesis on* “Software Defined GPS/Galileo Receiver” *April 2011.*
6. *K. Borre, D.M. Akos, N. Bertelsen, P. Rinder, and S.H. Jensen:* “A Software-Defined GPS and Galileo Receiver” *a single frequency approach* *(ISBN 0-8176-4390-7), © 2007 Birkhauser Boston*
7. *Charless Jefrrey* “An Introduction to GNSS” *GPS, GLONASS, Galileo and other global navigation satellite systems, first edition, NovAtel Inc 2010.*
8. *Interface Control Document2002* “Russian Institute of Space Device Engineering. GLONASS”.
9. *Sophia Y. Zheng* “Signal acquisition and Tracking for A Software GPS Receiver” *February, 2005 Blacksburg, Virginia.*
10. *Lei Dong* “IF GPS Signal Simulator Development and Verification” *Alberta University of Calgary, December 2003.*

# Websites

1. <http://en.wikipedia.org/wiki/GLONASS>
2. <http://www.scilab.org/>.
3. [www.insideGNSS.com](http://www.insideGNSS.com).
4. [www.navipedia.net](http://www.navipedia.net).
5. [www.ion.net](http://www.ion.net).
6. <http://www.geomatics.ucalgary.ca/>
7. <http://www.glonass-ianc.rsa.ru.//> source code available
8. [www.gpsworld.com](http://www.gpsworld.com).
9. [www.novatel.com](http://www.novatel.com).
10. www.gnss-sdr.ru

# APPENDIX

/\*This code is developed by A Jude Daniel Sunil, M.Pragna, K.Sravya, M.Swathi under the guidance of Dr.P.Laxminayarana,Principal Scientist (NERTU,OU) by 05/05/2014\*/

**C Code for functions and structures used in time synchronous pseudo code of Acquisition :**

**HEADER FILES**

**Settings.h**

#define NO\_OF\_SATLLITES 14

typedef struct

{

unsigned int msToProcess;

unsigned int numberOfChannels;

unsigned long skipNumberOfBytes;

unsigned short fileType;

double IF;

double samplingFreq;

double codeFreqBasis;

unsigned int codeLength;

unsigned short skipAcquisition;

unsigned short acqSearchBand;

double acqThreshold;

unsigned short acqSatelliteList[13];

char datatype;

}Settings;

void initSettings();

**generateCAcode.h**

void generateCAcode(short \*);

**complex.h**

typedef struct

{

double real;

double Imag;

} COMPLEX;

double c\_abs(COMPLEX \*x);

double c\_power(COMPLEX \*x);

double c\_angle(COMPLEX \*x);

void c\_add(COMPLEX \*x, COMPLEX \* y, COMPLEX \* z);

void c\_sub(COMPLEX \*x, COMPLEX \* y, COMPLEX \* z);

void c\_addsignals(COMPLEX \*x, COMPLEX \* y, COMPLEX \* z);

void c\_mult(COMPLEX \*x, COMPLEX \* y, COMPLEX \* z);

void c\_multsignals(COMPLEX \*x, COMPLEX \* y, COMPLEX \* z);

void conjugate(COMPLEX \*x,COMPLEX \*y);

void c\_absolute(COMPLEX \*x,double \*y);

**makeCAtable.h**

#include "complex.h"

void makeCAtable(short \*, COMPLEX \*);

**fft.h**

#include"complex.h"

void bitrev(COMPLEX \* X, COMPLEX \* Y, int N);

void fft\_ifft(COMPLEX \*p2x, COMPLEX \*p2Y, int N,int select);

**acqresults.h**

#define NO\_OF\_SATLLITES 14

typedef struct

{

double carrfreq;

double codePhase;

double peakMetric;

}acqresults;

**SOURCE FILES**

**Settings.c**

#include "settings.h"

extern Settings s1;

void initSettings()

{

int i;

//s1.acqThreshold = 5.9;

s1.msToProcess=34000;

s1.numberOfChannels=7;

s1.skipNumberOfBytes=16000000;

s1.fileType=2;

s1.IF=-3500000;

s1.samplingFreq=16000000;

s1.codeFreqBasis=511000;

s1.codeLength=511;

s1.skipAcquisition=0;

s1.acqSearchBand=14;

s1.acqThreshold=5;

for(i=1;i<=NO\_OF\_SATLLITES;i++)

{

s1.acqSatelliteList[i]=i;

}

s1.datatype = sizeof(char);

}

**generateCAcode.c**

#include "generateCAcode.h"

void generateCAcode(short \* p2CAcode)

{

int i,reg[9]={-1,-1,-1,-1,-1,-1,-1,-1,-1},j,temp;

for(i=0;i<511;i++)

{

temp=reg[4]\*reg[8];

p2CAcode[i]=-reg[6];

for(j=8;j>=1;j--)

{

reg[j]=reg[j-1];

}

reg[0]=temp;

}

}

**complex.c**

#include "complex.h"

#include "math.h"

#include "settings.h"

extern unsigned int samplesPerCode;

extern Settings s1;

extern int FFTLength;

double c\_abs(COMPLEX \*x)

{

double z;

z= sqrt(x->real \* x->real + x->Imag \* x->Imag);

return z;

}

double c\_power(COMPLEX \*x)

{

double z;

z= x->real \* x->real + x->Imag \* x->Imag;

return z;

}

double c\_angle(COMPLEX \*x)

{

double z;

z = atan(x->Imag/x->real);

return z;

}

void c\_add(COMPLEX \*x, COMPLEX \* y, COMPLEX \* z)

{

z->real = x->real + y->real;

z->Imag = x->Imag + y->Imag;

}

void c\_sub(COMPLEX \*x, COMPLEX \* y, COMPLEX \* z)

{

z->real = x->real - y->real;

z->Imag = x->Imag - y->Imag;

}

void c\_addsignals(COMPLEX \*x, COMPLEX \* y, COMPLEX \* z)

{

int i;

samplesPerCode = floor(s1.samplingFreq /(s1.codeFreqBasis /s1.codeLength));

for(i=0;i<5\*samplesPerCode;i++)

{

z[i].real = x[i].real - y[i].Imag;

z[i].Imag = x[i].Imag + y[i].real;

}

}

void c\_mult(COMPLEX \*x, COMPLEX \* y, COMPLEX \* z)

{

z->real = x->real \* y->real - x->Imag \* y->Imag;

z->Imag = x->real \* y->Imag + x->Imag \* y->real;

}

void conjugate(COMPLEX \*x,COMPLEX \*y)

{

int i;

FFTLength=pow(2,ceil(logf(5\*samplesPerCode)/logf(2)));

for(i=0;i<FFTLength;i++)

{

y[i].real=x[i].real;

y[i].Imag=-(x[i].Imag);

}

}

void c\_multsignals(COMPLEX \*x, COMPLEX \* y, COMPLEX \* z)

{

int i;

FFTLength=pow(2,ceil(logf(5\*samplesPerCode)/logf(2)));

for(i=0;i<FFTLength;i++)

{

z[i].real = x[i].real \* y[i].real - x[i].Imag \* y[i].Imag;

z[i].Imag = x[i].real \* y[i].Imag + x[i].Imag \* y[i].real;

}

}

void c\_absolute(COMPLEX \*x,double \*y)

{

int i;

FFTLength=pow(2,ceil(logf(5\*samplesPerCode)/logf(2)));

for(i=0;i<FFTLength;i++)

{

y[i]= (x[i].real \* x[i].real + x[i].Imag \* x[i].Imag);

}

}

**makeCAtable.c**

#include "settings.h"

#include "generateCAcode.h"

#include <math.h>

#include <stdlib.h>

#include <stdio.h>

#include "complex.h"

extern Settings s1;

void makeCAtable(short \*p2caCode, COMPLEX \* p2LocalcaCode);

void makeCAtable(short \*p2caCode, COMPLEX \* p2LocalcaCode)

{

short \* codeValueIndex,x;

unsigned short i,j;

double ts,tc;

int FFTLength, samplesPerCode;

samplesPerCode = floor(s1.samplingFreq /(s1.codeFreqBasis /s1.codeLength));

//FFTLength=pow(2,ceil(logf(samplesPerCode)/logf(2)));

codeValueIndex = (short \*) calloc(samplesPerCode,sizeof(short));

ts = 1/s1.samplingFreq;

tc = 1/s1.codeFreqBasis;

for(i=0;i<samplesPerCode;i++)

{

codeValueIndex[i] = ceil(i\*s1.codeFreqBasis/s1.samplingFreq);

if(i==0)

codeValueIndex[i] = 1;

}

for(i=0;i<samplesPerCode;i++)

{

if (codeValueIndex[i]==0)

i=i;

p2LocalcaCode[i].real =(double)p2caCode[codeValueIndex[i]-1];

p2LocalcaCode[i].Imag = 0;

}

}

**fft.c**

#include "stdlib.h"

#include "math.h"

#include "fft.h"

#include <stdio.h>

#define PI 3.141592653589793

void bitrev(COMPLEX \* p2X, COMPLEX \*p2Y, int N)

{

unsigned int n,p,q,l,Nbits;

Nbits=ceil(logf(N)/logf(2.0));

for (l=0;l<N; l++)

{

q=l;

p=0;

for (n=0;n<Nbits;n++)

{

p=(p<<1)+ (q & 1);

q=q>>1;

}

p2Y[p].real = p2X[l].real;

p2Y[p].Imag = p2X[l].Imag;

}

}

void fft\_ifft(COMPLEX \*p2x, COMPLEX \*p2Y, int N,int select)

{

int NStages,stage,NpointFFT,Nffts,Nthfft,Nbutterflys,nthButterfly,InputsIndex[2];

COMPLEX \*W,WW,temp,xinput[2],xoutput[2];

int i,j;

// FILE \*fp;

NStages=ceil(logf(N)/logf(2.0));

bitrev(p2x, p2Y, N);

W=(COMPLEX\*) malloc(N/2\*sizeof(COMPLEX));

for(i=0;i<N/2;i++)

{

W[i].real=cosf(-2\*select\*PI\*i/N);

W[i].Imag=sinf(-2\*select\*PI\*i/N);

}

for(stage=1;stage<=NStages;stage++)

{

NpointFFT=powf((double)2,(double)stage);

Nffts=N/NpointFFT;

for(Nthfft=0;Nthfft<Nffts;Nthfft++)

{

Nbutterflys=NpointFFT/2;

for(nthButterfly=0;nthButterfly<Nbutterflys;nthButterfly++)

{

j=powf((double)2, (double)(NStages-stage))\*nthButterfly;

WW.Imag=W[j].Imag;

WW.real=W[j].real;

InputsIndex[0]=Nthfft\*NpointFFT+nthButterfly+0;

InputsIndex[1] = Nthfft \* NpointFFT + nthButterfly + NpointFFT/2;

xinput[0].real = p2Y[InputsIndex[0]].real;

xinput[0].Imag = p2Y[InputsIndex[0]].Imag;

xinput[1].real = p2Y[InputsIndex[1]].real;

xinput[1].Imag = p2Y[InputsIndex[1]].Imag;

c\_mult(&p2Y[InputsIndex[1]],&WW,&xinput[1]);

c\_add(&xinput[0],&xinput[1],&xoutput[0]);

c\_sub(&xinput[0],&xinput[1],&xoutput[1]);

for(i=0;i<2;i++)

{

p2Y[InputsIndex[i]].real = xoutput[i].real;

p2Y[InputsIndex[i]].Imag = xoutput[i].Imag;

}

}

}

}

if(select==-1)

for(i=0;i<N;i++)

{

p2Y[i].Imag/=N;

p2Y[i].real/=N;

}

free(W);

}

**C Code for functions and structures used in time synchronous pseudo code of Tracking :**

**HEADER FILES**

**acqResults.h**

struct acqResults

{

double carrFreq;

double codePhase;

double peakMetric;

}acqresults[14];

void AcqResults();

**settings.h**

#define NO\_OF\_SATLLITES 14

#define MsToProcess 34000

typedef struct

{

unsigned int msToProcess;

unsigned int numberOfChannels;

unsigned long skipNumberOfBytes;

unsigned short fileType;

double IF;

double samplingFreq;

double codeFreqBasis;

unsigned int codeLength;

unsigned short skipAcquisition;

unsigned short acqSearchBand;

double acqThreshold;

char acqCohIntegration;

float dllDampingRatio;

char dllNoiseBandwidth;

float dllCorrelatorSpacing;

float pllDampingRatio;

char pllNoiseBandwidth;

int navSolPeriod;

char elevationMask;

char useTropCorr;

char plotTracking;

double c;

float startOffset;

unsigned short acqSatelliteList[13];

char datatype;

}Settings;

void initSettings();

**preRun.h**

#include "settings.h"

#include"acqResults.h"

extern Settings s1;

struct channel

{

short PRN;

double acquiredFreq;

float codePhase;

char status;

}Channel[7];

void preRun(struct acqResults acqresults[],Settings s1);

**showChannelStatus.h**

#include"preRun.h"

void showChannelStatus(struct channel ch[],Settings s1);

**calcLoopCoef.h**

float\* calcLoopCoef(char,float,float);

**generateCAcode.h**

short\* generateCAcode(short \*);

**complex.h**

typedef struct

{

double real;

double Imag;

} COMPLEX;

double c\_abs(COMPLEX \*x);

double c\_power(COMPLEX \*x);

double c\_angle(COMPLEX \*x);

void c\_add(COMPLEX \*x, COMPLEX \* y, COMPLEX \* z);

void c\_sub(COMPLEX \*x, COMPLEX \* y, COMPLEX \* z);

**initFeddBackTrack.h**

#include"showChannelStatus.h"

struct Feedback\_Parameters

{

double codeFreq;

double remCodePhase;

double remCarrPhase;

double oldCodeNco;

double oldCodeError;

double oldCarrNco;

double oldCarrError;

double I\_P;

double I\_E;

double I\_L;

double Q\_P;

double Q\_E;

double Q\_L;

double pllDiscrFilt;

double pllDiscr;

double dllDiscrFilt;

double dllDiscr;

double carrFreq;

double carrFreqBasis;

double absoluteSample;

short PRN;

}Feedback\_parameters[7];

struct Feedback\_Parameters \* initFeddBackTrack(Settings s1,struct channel Ch[]);

**initFixedParameters.h**

#include "initFeddBackTrack.h"

struct FixedParameters

{

float earlyLateSpc;

float PDIcode;

float tau1code;

float tau2code;

float PDIcarr;

float tau1carr;

float tau2carr;

}fixedParameters[7];

struct FixedParameters \*initFixedParameters(Settings s1);

**trackIter.h**

#include<stdio.h>

#include"initFixedParameters.h"

struct Feedback\_Parameters TrackIter(FILE \*,struct Feedback\_Parameters FB,struct FixedParameters FP,Settings s1,struct channel ch[]);

**initTrackResults.h**

#include"trackIter.h"

struct trackResults

{

char status;

int absoluteSample[MsToProcess];

double codeFreq[MsToProcess];

double carrFreq[MsToProcess];

double I\_P[MsToProcess];

double I\_E[MsToProcess];

double I\_L[MsToProcess];

double Q\_E[MsToProcess];

double Q\_P[MsToProcess];

double Q\_L[MsToProcess];

double dllDiscr[MsToProcess];

double dllDiscrFilt[MsToProcess];

double pllDiscr[MsToProcess];

double pllDiscrFilt[MsToProcess];

short PRN;

}trackresults[7];

struct trackResults \*initTrackResults(Settings s1);

**SOURCE FILES**

**acqResults.c**

#include"acqResults.h"

#include<stdio.h>

void AcqResults()

{

int i;

float j;

FILE \*fptestpm,\*fptestcf,\*fptestcp;

fptestcf = fopen("C:\\Users\\home\\Desktop\\Results\\Cresults\\testvarcf.txt","rt");

fptestcp = fopen("C:\\Users\\home\\Desktop\\Results\\Cresults\\testvarcp.txt","rt");

fptestpm = fopen("C:\\Users\\home\\Desktop\\Results\\Cresults\\testvarpm.txt","rt"); for(i=0;i<14;i++)

{

fscanf(fptestcf,"%f",&j);

acqresults[i].carrFreq = j;

fscanf(fptestcp,"%f",&j);

acqresults[i].codePhase = j;

fscanf(fptestpm,"%f",&j);

acqresults[i].peakMetric = j;

}

fclose(fptestcf);

fclose(fptestcp);

fclose(fptestpm);

}

**Settings.c**

#include "settings.h"

extern Settings s1;

void initSettings()

{

int i;

s1.msToProcess=34000;

s1.numberOfChannels=7;

s1.skipNumberOfBytes=16000000;

s1.fileType=2;

s1.IF=-3500000;

s1.samplingFreq=16000000;

s1.codeFreqBasis=511000;

s1.codeLength=511;

s1.skipAcquisition=0;

s1.acqSearchBand=14;

s1.acqThreshold=5;

s1.dllDampingRatio=0.7;

s1.dllNoiseBandwidth=2;

s1.dllCorrelatorSpacing=0.5;

s1.pllDampingRatio=0.7;

s1.pllNoiseBandwidth=25;

s1.navSolPeriod=500;

s1.elevationMask=0;

s1.useTropCorr=1;

s1.plotTracking=1;

s1.c=299792458;

s1.startOffset=68.802;

for(i=1;i<=NO\_OF\_SATLLITES;i++)

{

s1.acqSatelliteList[i]=i;

}

s1.datatype = sizeof(char);

}

**preRun.c**

#include "preRun.h"

#include<stdio.h>

void preRun(struct acqResults acqresults[],Settings s1)

{

unsigned int noOfVisSat,min;

int i,a,j,PRNindexes[14],b,c;

double peakMetric[14],swap,junk[14],carrFreq[14],codePhase[14];

FILE \*fptest5;

for(i=0;i<NO\_OF\_SATLLITES;i++)

{

Channel[i].acquiredFreq = 0;

Channel[i].codePhase = 0;

Channel[i].PRN = 0;

Channel[i].status = '-';

}

for(i=0;i<14;i++)

junk[i] = acqresults[i].peakMetric;

for (i = 0 ; i < 14; i++)

{

for (j = 0 ; j < 14 - i - 1; j++)

{

if (junk[j] < junk[j+1]) /\* For decreasing order use < \*/

{

swap = junk[j];

junk[j] = junk[j+1];

junk[j+1] = swap;

}

}

}

for(i=0;i<14;i++)

{

for(j=0;j<14;j++)

{

if(junk[i] == acqresults[j].peakMetric)

PRNindexes[i] = j;

}

}

noOfVisSat = 0;

for(i=0;i<14;i++)

{

if(acqresults[i].carrFreq!=0)

noOfVisSat++;

}

if(s1.numberOfChannels < noOfVisSat)

min = s1.numberOfChannels;

else

min = noOfVisSat;

for(i=0;i<min;i++)

{

Channel[i].PRN = PRNindexes[i];

Channel[i].acquiredFreq = acqresults[PRNindexes[i]].carrFreq;

Channel[i].codePhase = acqresults[PRNindexes[i]].codePhase;

Channel[i].status = 'T';

}

}

**showChannelStatus.c**

#include "showChannelStatus.h"

#include<stdio.h>

void showChannelStatus(struct channel ch[],Settings s1)

{

int channelNr;

float a;

printf("\n\*=========\*=====\*===============\*===========\*=============\*======\*\n");

printf("| Channel | PRN | Frequency | Doppler | Code Offset | Status |\n");

printf("\*=========\*=====\*===============\*===========\*=============\*========\*\n");

for(channelNr=0; channelNr<s1.numberOfChannels;channelNr++)

{

a = ch[channelNr].acquiredFreq - (s1.IF +(((ch[channelNr].PRN)+1)\*(562500)));

if (ch[channelNr].status != '-')

printf("| %2d | %3d | %12.0f | %5.0f | %6f | %1c |\n",channelNr,ch[channelNr].PRN,ch[channelNr].acquiredFreq,a,ch[channelNr].codePhase,ch[channelNr].status);

else

printf("| %2d | --- | ------------ | ----- | ------ | Off |\n",channelNr);

}

printf("\*=========\*=====\*===============\*===========\*=============\*========\*\n\n");

}

**calcLoopCoef.c**

#include<stdlib.h>

#include<math.h>

float\* calcLoopCoef(char LBW,float Zeta,float k)

{

float Wn;

float \* tau = (float \*)calloc(2,sizeof(float));

Wn = LBW\*8\*Zeta / (4\* pow(Zeta,2) + 1);

\*tau = k / (Wn \* Wn);

\*++tau = 2.0 \* Zeta / Wn;

return(--tau);

}

**generateCAcode.c**

#include "generateCAcode.h"

#include<stdio.h>

short\* generateCAcode(short \*p2CAcode)

{

int i,reg[9]={-1,-1,-1,-1,-1,-1,-1,-1,-1},j,temp;

for(i=0;i<511;i++)

{

temp=reg[4]\*reg[8];

p2CAcode[i]=-reg[6];

for(j=8;j>=1;j--)

{

reg[j]=reg[j-1];

}

reg[0]=temp;

}

return p2CAcode;

}

**complex.c**

#include "complex.h"

#include "math.h"

double c\_abs(COMPLEX \*x)

{

double z;

z= sqrt(x->real \* x->real + x->Imag \* x->Imag);

return z;

}

double c\_power(COMPLEX \*x)

{

double z;

z= x->real \* x->real + x->Imag \* x->Imag;

return z;

}

double c\_angle(COMPLEX \*x)

{

double z;

z = atan(x->Imag/x->real);

return z;

}

void c\_add(COMPLEX \*x, COMPLEX \* y, COMPLEX \* z)

{

z->real = x->real + y->real;

z->Imag = x->Imag + y->Imag;

}

void c\_sub(COMPLEX \*x, COMPLEX \* y, COMPLEX \* z)

{

z->real = x->real - y->real;

z->Imag = x->Imag - y->Imag;

}

**initFeddBackTrack.c**

#include"initFeddBackTrack.h"

#include<stdio.h>

struct Feedback\_Parameters \* initFeddBackTrack(Settings s1,struct channel Ch[])

{

struct Feedback\_Parameters Track[7];

char dataAdaptCoeff,channelNr,i;

FILE \*fptest5;

for(i=0;i<s1.numberOfChannels;i++)

{

Track[i].codeFreq = s1.codeFreqBasis;

//define residual code phase (in chips)

Track[i].remCodePhase = 0.0;

//define residual carrier phase

Track[i].remCarrPhase = 0.0;

//code tracking loop parameters

Track[i].oldCodeNco = 0.0;

Track[i].oldCodeError = 0.0;

//carrier/Costas loop parameters

Track[i].oldCarrNco = 0.0;

Track[i].oldCarrError = 0.0;

Track[i].I\_P = 0;

Track[i].I\_E = 0;

Track[i].I\_L = 0;

Track[i].Q\_P = 0;

Track[i].Q\_E = 0;

Track[i].Q\_L = 0;

Track[i].pllDiscrFilt = 0;

Track[i].pllDiscr = 0;

Track[i].dllDiscrFilt = 0;

Track[i].dllDiscr = 0;

}

if (s1.fileType==1)

dataAdaptCoeff=1;

else

dataAdaptCoeff=2;

//define carrier frequency which is used over whole tracking period

for(channelNr=0;channelNr<s1.numberOfChannels;channelNr++)

{

Track[channelNr].carrFreq = Ch[channelNr].acquiredFreq;

Track[channelNr].carrFreqBasis = Ch[channelNr].acquiredFreq;

Track[channelNr].absoluteSample= dataAdaptCoeff\*(s1.skipNumberOfBytes + (Ch[channelNr].codePhase+1)-1);

}

for(i=0;i<s1.numberOfChannels;i++)

{

Feedback\_parameters[i] = Track[i];

}

return(&Feedback\_parameters[0]);

}

**initFixedParameters.c**

#include"initFixedParameters.h"

#include"calcLoopCoef.h"

struct FixedParameters \*initFixedParameters(Settings s1)

{

//float tau1Code,tau2Code,tau1Carr,tau2Carr;

float \*taucode,\*taucarr;

char i;

for(i=0;i<s1.numberOfChannels;i++)

{

fixedParameters[i].earlyLateSpc = s1.dllCorrelatorSpacing;

//Summation interval

fixedParameters[i].PDIcode = 0.001;

//Calculate filter coefficient values

taucode = calcLoopCoef(s1.dllNoiseBandwidth,s1.dllDampingRatio,1.0);

fixedParameters[i].tau1code = \*taucode;

fixedParameters[i].tau2code = \*++taucode;

//--- PLL variables --------------------------------------------------------

//Summation interval

fixedParameters[i].PDIcarr = 0.001;

//Calculate filter coefficient values

taucarr = calcLoopCoef(s1.pllNoiseBandwidth,s1.pllDampingRatio,0.25);

fixedParameters[i].tau1carr = \*taucarr;

fixedParameters[i].tau2carr = \*++taucarr;

}

return fixedParameters;

}

**trackIter.c**

#include"trackIter.h"

#include"complex.h"

#include<math.h>

#include<stdlib.h>

long double PI=3.141592653589793;

extern Settings s1;

extern short \*\*CAcode;

extern z;

struct Feedback\_Parameters TrackIter(FILE \*fp,struct Feedback\_Parameters FB,struct FixedParameters FP,Settings s1,struct channel ch[])

{

double codePhaseStep, blksize, k, \*tcode, \*tcode2, \*earlyCode, \*lateCode, \*promptCode, \*time, \*iBasebandSignal,I\_E = 0,Q\_E = 0,I\_P = 0,Q\_P = 0,I\_L = 0,Q\_L = 0,codeNco,carrNco,codeError,carrError;

FILE \*fptest5;

long double \*qBasebandSignal,\*trigarg;

float samplesRead;

short i,j,\*caCode;

char dataAdaptCoeff,\*data,\*rawsignal1,\*rawsignal2,\*p2data;

COMPLEX \*rawsignal;

caCode = (short \*)calloc(513,sizeof(short));

tcode = (double \*)calloc(16050,sizeof(double));

tcode2 = (double \*)calloc(16050,sizeof(double));

earlyCode = (double \*)calloc(16050,sizeof(double));

lateCode = (double \*)calloc(16050,sizeof(double));

promptCode = (double \*)calloc(16050,sizeof(double));

time = (double \*)calloc(16050,sizeof(double));

trigarg = (long double \*)calloc(16050,sizeof(long double));

qBasebandSignal = (long double \*)calloc(16050,sizeof(long double));

iBasebandSignal = (double \*)calloc(16050,sizeof(double));

data = (char \*)calloc(32100,sizeof(char));

rawsignal = (COMPLEX \*)calloc(16500,sizeof(COMPLEX));

if(rawsignal==NULL)

printf("insufficient memory");

i = FB.PRN;

for(j = 1;j < 512;j++)

{

caCode[j] = CAcode[i][j-1];

}

//Then make it possible to do early and late versions

caCode[0] = CAcode[i][510];

caCode[512] = CAcode[i][0];

codePhaseStep = FB.codeFreq / s1.samplingFreq;

// Update the phasestep based on code freq (variable) and

//sampling freq uency (fixed)

blksize = ceil((s1.codeLength-FB.remCodePhase) / codePhaseStep);

if (s1.fileType==1)

dataAdaptCoeff=1;

else

dataAdaptCoeff=2;

// Read in the appropriate number of samples to process this interation

fseek(fp,FB.absoluteSample,0);

samplesRead = fread(data,1,dataAdaptCoeff\*blksize,fp);

p2data = data;

FB.absoluteSample = ftell(fp);

if (dataAdaptCoeff==2)

{

for(i = 0;i < (samplesRead/2);i++)

{

rawsignal[i].real = (double) \*p2data++;

rawsignal[i].Imag = (double) \*p2data++;

}

}

if (samplesRead != dataAdaptCoeff\*blksize)

{

printf("Not able to read the specified number of samples for tracking, exiting");

fclose(fp);

return FB;

}

for(k = (FB.remCodePhase-FP.earlyLateSpc),j=0;k <=(((blksize-1) \*codePhaseStep + FB.remCodePhase+FP.earlyLateSpc)+1) && j<16000;k+=codePhaseStep,j++)

{

tcode[j] = k;

tcode2[j] = ceil(tcode[j]) + 1;

earlyCode[j] = caCode[(int)(tcode2[j]-1)];

}

for(k = (FB.remCodePhase+FP.earlyLateSpc),j=0;(k <= ((blksize-1) \*codePhaseStep + FB.remCodePhase+FP.earlyLateSpc)+1) && j<16000;k+=codePhaseStep,j++)

{

tcode[j] = k;

tcode2[j] = ceil(tcode[j]) + 1;

lateCode[j] = caCode[(int)(tcode2[j]-1)];

}

for(k = (FB.remCodePhase),j=0;(k <= (((blksize-1)\*codePhaseStep+FB.remCodePhase)+1)) ; k+=codePhaseStep,j++)

{

tcode[j] = k;

tcode2[j] = ceil(tcode[j]) + 1;

promptCode[j] = caCode[(int)(tcode2[j]-1)];

}

FB.remCodePhase = tcode[(int)(blksize-1)] + codePhaseStep - 511.0;

//Generate the carrier frequency to mix the signal to baseband -----------

// Get the argument to sin/cos functions

for(k = 0;k < blksize;k++)

{

time[(int)k] = k/s1.samplingFreq;

trigarg[(int)k] = ((FB.carrFreq\* 2.0 \* PI) \* time[(int)k]) + FB.remCarrPhase;

// Generate the six standard accumulated values ---------------------------

// First mix to baseband

qBasebandSignal[(int)k] = (cos(trigarg[(int)k])\*rawsignal[(int)k].real ) - (sin(trigarg[(int)k])\*rawsignal[(int)k].Imag) ;

iBasebandSignal[(int)k] = (cos(trigarg[(int)k])\*rawsignal[(int)k].Imag ) + (sin(trigarg[(int)k])\*rawsignal[(int)k].real) ;

//Now get early, late, and prompt values for each

I\_E = I\_E + (earlyCode[(int)k] \* iBasebandSignal[(int)k]);

Q\_E = Q\_E + (earlyCode[(int)k] \* qBasebandSignal[(int)k]);

I\_P = I\_P + (promptCode[(int)k] \* iBasebandSignal[(int)k]);

Q\_P = Q\_P + (promptCode[(int)k] \* qBasebandSignal[(int)k]);

I\_L = I\_L + (lateCode[(int)k] \* iBasebandSignal[(int)k]);

Q\_L = Q\_L + (lateCode[(int)k] \* qBasebandSignal[(int)k]);

}

if(rawsignal !=NULL)

free(rawsignal);

time[16000] = k/s1.samplingFreq;

trigarg[16000] = ((FB.carrFreq \* 2.0 \* PI) \* time[16000]) + FB.remCarrPhase;

FB.remCarrPhase = (trigarg[(int)(blksize)]-(int) (trigarg[(int)(blksize)]/((2 \* PI)))\*(2 \* PI));

// Find PLL error and update carrier NCO ----------------------------------

//Implement carrier loop discriminator (phase detector)

carrError = atan(Q\_P / I\_P) / (2.0 \* PI);

//Implement carrier loop filter and generate NCO command

carrNco = FB.oldCarrNco + (FP.tau2carr/FP.tau1carr) \* (carrError - FB.oldCarrError) + carrError \* (FP.PDIcarr/FP.tau1carr);

FB.oldCarrNco = carrNco;

FB.oldCarrError = carrError;

//Modify carrier freq based on NCO command

FB.carrFreq = FB.carrFreqBasis + carrNco;

// Find DLL error and update code NCO -------------------------------------

codeError = (sqrt(I\_E \* I\_E + Q\_E \* Q\_E) - sqrt(I\_L \* I\_L + Q\_L \* Q\_L)) / (sqrt(I\_E \* I\_E + Q\_E \* Q\_E) + sqrt(I\_L \* I\_L + Q\_L \* Q\_L));

// Implement code loop filter and generate NCO command

codeNco = FB.oldCodeNco + (FP.tau2code/FP.tau1code) \* (codeError - FB.oldCodeError) + codeError \* (FP.PDIcode/FP.tau1code);

FB.oldCodeNco = codeNco;

FB.oldCodeError = codeError;

FB.dllDiscr = FB.oldCodeError;

FB.dllDiscrFilt = FB.oldCodeNco;

FB.pllDiscr = FB.oldCarrError;

FB.pllDiscrFilt = FB.oldCarrNco;

// Modify code freq based on NCO command

FB.codeFreq = s1.codeFreqBasis - codeNco;

FB.I\_P = I\_P;

FB.I\_E = I\_E;

FB.I\_L = I\_L;

FB.Q\_P = Q\_P;

FB.Q\_E = Q\_E;

FB.Q\_L = Q\_L;

free(caCode);

free(tcode);

free(tcode2);

free(earlyCode);

free(lateCode);

free(promptCode);

free(time);

free(trigarg);

free(qBasebandSignal);

free(iBasebandSignal);

free(data);

return FB;

}

**initTrackResults.c**

#include"initTrackResults.h"

struct trackResults \*initTrackResults(Settings s1)

{

int i,j;

for(i=0;i<s1.numberOfChannels;i++)

{

trackresults[i].status = '-'; //No tracked signal, or lost lock

trackresults[i].PRN = 0;

for(j=0;j<s1.msToProcess;j++)

{

//The absolute sample in the record of the C/A code start:

trackresults[i].absoluteSample[j] = 0;

//Freq of the C/A code:

trackresults[i].codeFreq[j] = 0;

//Frequency of the tracked carrier wave:

trackresults[i].carrFreq[j] = 0;

// Outputs from the correlators (In-phase):

trackresults[i].I\_P[j] = 0;

trackresults[i].I\_E[j] = 0;

trackresults[i].I\_L[j] = 0;

// Outputs from the correlators (Quadrature-phase):

trackresults[i].Q\_E[j] = 0;

trackresults[i].Q\_P[j] = 0;

trackresults[i].Q\_L[j] = 0;

// Loop discriminators

trackresults[i].dllDiscr[j] = 0;

trackresults[i].dllDiscrFilt[j] = 0;

trackresults[i].pllDiscr[j] = 0;

trackresults[i].pllDiscrFilt[j] = 0;

}

}

return trackresults;

}