1-1-2008

A system level implementation of wavelet based filtering for GNSS signals

Jorge Leon
Ryerson University

A SYSTEM LEVEL IMPLEMENTATION OF WAVELET BASED FILTERING FOR GNSS SIGNALS

by

Jorge Leon

B.Sc., Queen's University, 2002

A project submitted in partial fulfillment of the
requirements for the degree of

Master of Engineering in the Program of Electrical
and Computer Engineering

Ryerson University

Toronto, Ontario, Canada, 2008

AUTHOR'S DECLARATION

I hereby declare that I am the sole author of this project.

I authorize Ryerson University to lend this project to other institutions or individuals for the purpose of scholarly research.

Signature:

I further authorize Ryerson University to reproduce this project by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Signature:

ABSTRACT

A System Level Implementation of Wavelet Based Filtering For GNSS Signals

Degree of Master of Engineering in the Program of Electrical and Computer Engineering

Ryerson University, 2008

By Jorge Leon

A project is presented to study the Global Positioning System and learn how to apply wavelet analysis to mitigate the effects of multipath errors on GNSS signals. The analysis is carried out using the SystemC language to demonstrate how one may try to implement the GPS signal wavelet filter in hardware. Wavelet analysis, the SystemC library and additional tools are discussed in detail. Design issues such as control signaling and position estimation are explained. System evaluation is performed at two levels, one using cross correlation of signals and the second by measuring the amount of clustering in position plots.

TABLE OF CONTENTS

LIST OF FIGURES ..... **II**

ACKNOWLEDGMENTS ..... **III**

GLOSSARY ..... **IV**

CHAPTER 1 1

Introduction ***1***

GPS Applications 1

A Streetcar GPS Application Perspective 2

GPS Overview 3

The Space Segment 4

The Control Segment 5

The User Segment 6

Multipath Interference 6

Current Approaches to Multipath Mitigation 8

Objective of the Project 9

Organization of the Project 9

CHAPTER 2 10

Implementation Tools ***10***

SystemC Modeling 10

Wavelet Analysis of GNSS Signals 12

GPS Position Estimation 17

GPS Toolkit 19

CHAPTER 3 21

Wavelet Filter Implementation ***21***

Wavelet Filter Module 22

Source Module 23

Sink Module 24

System Integration and Analysis 25

CHAPTER 4 27

Position Estimation Implementation ***27***

New Source Module 27

New Sink Module 28

System Evaluation 30

CHAPTER 5 36

Conclusions and Recommendations ***36***

BIBLIOGRAPHY **40**

APPENDIX **42**

LIST OF FIGURES

Number *Page*

1. Possible streetcar scheduling system implementation 2
2. GPS Satellites orbit the Earth on 6 orbital planes 4
3. Simplified satellite signal block diagram 5
4. SystemC 2.0 Language Architecture 12
5. Meyer and Daubechies db2 wavelets 13
6. Time-frequency tiles with Daubechies basis function 14
7. Three-level wavelet decomposition 16
8. Three-dimensional coordinate system 17
9. GPS filter system block diagram 21
10. System block diagram with signals 22
11. Cross-correlation between original and filtered signals 25
12. System signals between the three modules 26
13. Ideal position plot 31
14. Data set 1 before wavelet filtering 32
15. Data set 1 after wavelet filtering 32
16. Data set 2 before wavelet filtering 33
17. Data set 2 after wavelet filtering 33
18. System signals for position estimation implementation 35
19. System signals during Source-Filter communication 35

ACKNOWLEDGMENTS

I wish to express my sincere appreciation to Dr. Sri Krishnan and Dr. Alagan Anpalagan for their guidance during my research and assistance in the preparation of this project. In addition, I would like to thank to Dr. Gul Khan whose technical advice was helpful during the early programming phase of this undertaking.

Thank you.

Jorge Leon

GLOSSARY

BPSK - Binary Phase Shift Keying

DWT - Discrete Wavelet Transform

ECEF - Earth Centered, Earth Fixed coordinate system

GLONASS – Global'naya Navigatsionnaya Sputnikovaya Sistema (Russian GNSS)

GNSS - Global Navigation Satellite System

GPS - Global Positioning System

HDL - Hardware Description Languages

IDWT - Inverse Discrete Wavelet Transform

NAVSTAR - Navigation System with Timing and Ranging

PRN - Pseudo Random Noise

Pseudorange – The distance between a satellite and a GPS user, in meters

TLM - Transaction-Level Modeling

Trilateration - A method used to determine the relative position of objects using geometry of triangles. The solution is equivalent to the intersection of three spheres.

Chapter 1

INTRODUCTION

For thousands of years people have been trying to solve the problem of positioning and navigation. Human ingenuity and necessity have led various entities such as governments to construct a wide range of technologies and mechanisms to provide answers. All of these technologies, ranging from mechanical clocks and watches measuring components of astronomical time to radio signals such as radar and atomic clocks, have converged in the development of modern Global Navigation Satellite Systems (GNSS). GNSS are systems which uses a network of satellites for estimating position anywhere on the Earth. The first system to be deployed is the Navigation System with Timing and Ranging (NAVSTAR) Global Positioning System (GPS) consisting of 24 satellites (32 as of March 2008) which were placed into orbit by the U.S. Department of Defense in the 1970's. Other GNSS systems include GLONASS and Galileo being developed by Russia and the European Union, respectively. While all GNSS systems are implemented differently, the basic idea for calculation position is the same. As of August 2008, GPS is the only functioning system available.

GPS Applications

GPS was originally intended for military applications such as navigation, target tracking, missile guidance, search and rescue, and reconnaissance for map creation. In the 1980's, the government made the system available for civilian use. Since then the use of GPS systems have become more and more widespread and the number of applications has been rapidly growing. Some examples of GPS applications include mapping of a user's position on an interactive display, calculating routes, tracking devices or users, clock synchronization, and finding nearby services and businesses such as gas stations and restaurants.

A Streetcar GPS Application Perspective

Given these basic applications, more specific real-life applications can be defined. One application of particular interest is the accurate scheduling of streetcars in downtown Toronto. Using GPS receivers, the estimated time of arrival of a streetcar at a particular station can be determined given the current position of the streetcar, the speed of the streetcar and the location of the destination. The first two parameters can be calculated by the receivers using satellite information. The destination parameter would have to be predefined and entered into the system as part of the route of the streetcar. The distance between the streetcar current position and destination can be found using the route information. Finally, an estimate of the arrival time can be made by applying the basic velocity equation.

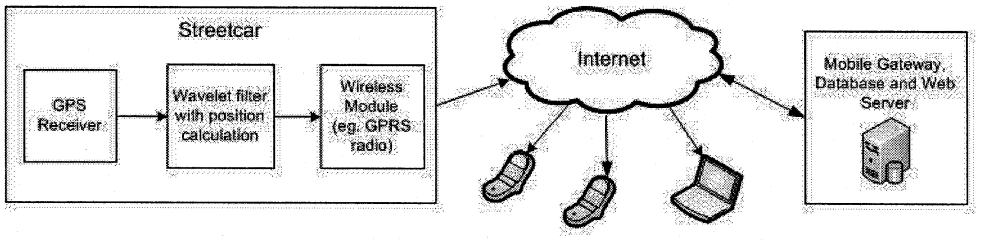

Figure 1: Possible streetcar scheduling system implementation

The streetcar scheduling system can be implemented in different ways. One possible implementation is shown in Figure 1. It consists of a GPS receiver, wavelet filter and wireless module mounted on streetcars. The GPS receiver is needed to obtain data from GPS satellites and provide pseudorange raw data to the wavelet filter module. The wavelet filter module obviously would implement the work presented in this project. The calculated, current and accurate position of the streetcar from this second module would be passed on to the wireless module for data to be sent wirelessly via a cellular data network to a mobile gateway server located at the TTC main office. The position information sent to the server would be accompanied by a streetcar identifier and a timestamp. The mobile

gateway server would have to be set up with a database containing the coordinates of each of the stations as well as a list of streetcars. The streetcar GPS component would have to periodically send their position to the server. The server collects and stores the information then computes the estimated time of arrival of the streetcar at its next stop. This information could then easily be posted on a website for user access.

The streetcar scheduling application is interesting because it can improve the quality of service provided by the Toronto Transit Commission (TTC). Commuters would be able to better schedule their activities and use their time more efficiently if they knew the precise time of arrival of a streetcar at a given station. The benefits would be more appreciable if the arrival time information was not only available at the station but also via the internet. Users could check online at home or through data enabled cellular phones and PDA's. In addition, the application could be further enhanced to keep track of fleet status and generate survey reports for route optimization.

In this streetcar application, precise arrival time estimates can only be achieved with accurate position values from a GPS receiver. However, the task of calculating accurate position coordinates becomes increasingly difficult in densely populated downtown areas with large buildings because of satellite signal errors caused by multipath and interference. Many different techniques for multipath and interference mitigation have been studied and applied to different systems. Traditional de-noising techniques are based on linear methods and include the popular Wiener filter [1]. In this project, attention is focused on applying a nonlinear method using wavelets for signal de-noising to help improve GPS position accuracy in a downtown environment.

GPS Overview

The Global Position System (GPS) uses a constellation of satellites transmitting microwave signals, which are used by GPS receivers to determine their location. The satellites continuously send out time stamped messages along with orbit information. Since

microwave signals travel at the speed of light, and the time between signals is known from the time stamps, the receiver can calculate the distance to the satellites. Using the orbit and distance information to at least 3 satellites, a user's location (longitude, latitude and altitude) can be found. To account for timing errors, a fourth satellite is used. Therefore, at least 4 satellites are required for normal GPS operation. In addition, the signals can also be used to calculate speed, direction and time. The GPS system is comprised of three functional segments: (1) the space segment, (2) the control segment, and (3) the user segment [6].

The Space Segment

The space segment consists of a constellation of satellites orbiting the Earth on 6 different orbital planes at a height of approximately 20,180 km above the Earth's surface. This is shown in Figure 2. The satellites orbit with an incline of 55° to the equator and take about 12 hours to complete their orbit. The satellite orbits are organized in such a way that communication with at least 4 satellites is ensured at all times throughout the world. The satellites continuously transmit signals which can be received anywhere on Earth. For a particular location, the signal from a given satellite will only be received during its effective range, before its line of sight is lost.

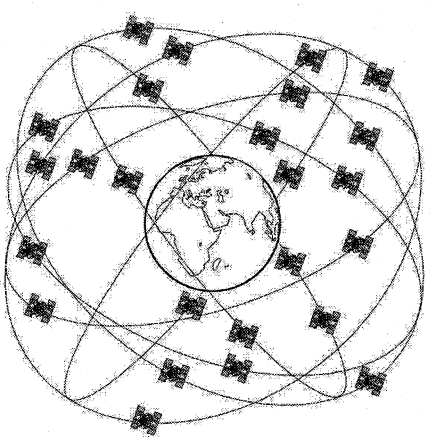

Figure 2: GPS Satellites orbit the Earth on 6 orbital planes

All satellites are equipped with atomic clocks for signal synchronization. Signals are transmitted on two carrier frequencies known as L1 and L2, having values of 1575.42 MHz and 1227.60 MHz respectively. Data is modulated using a C/A (Coarse/Acquisition) code which uses a uniquely assigned Pseudo Random Noise (PRN) code for each satellite. The data in turn modulates the L1 carrier using Binary Phase Shift Keying (BPSK). A basic block diagram of these signals is shown in Figure 3. After the antenna gain, the effective radiated power from the satellite is 26.8 dBW. The received signal strength received on Earth is approximately -160 dBW. In addition to time and synchronization information, the signals also carry precise orbital data (ephemeris), time correction information, approximate orbital data for all satellites (almanac), correction signals to calculate signal transit time, data on the ionosphere, and information on the operating status (health) of the satellite.

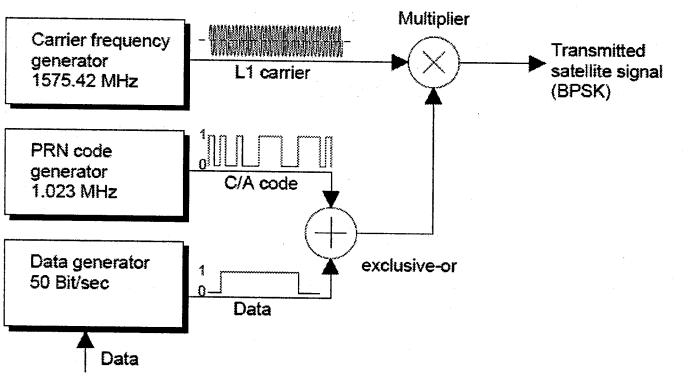

Figure 3: Simplified satellite signal block diagram

The Control Segment

The GPS control segment is responsible for tracking satellite orbital paths from various control stations on Earth. Navigational updates are sent regularly from the control stations in order to synchronize the satellite's onboard atomic clocks and adjust the ephemeris found in the orbital model used by the satellite. The control segment also relays the

almanac and orbital data from other satellites. It also monitors and controls the satellite health.

The User Segment

The user segment consists of GPS receiver devices which consist of an antenna, a receiver-processor and a very stable clock. The receiver processes the signals sent by the satellites by first separating the time signals according to the corresponding C/A code, effectively identifying each satellite. The pseudorange of the available satellites is then calculated. The almanac which stores satellite locations is then used in conjunction with the satellite pseudoranges to compute the position. The procedure used to make this calculation is called *trilateration*. This procedure is equivalent to finding the intersection of spheres.

Depending on the application, the receiver's user interface and extra processing will vary. While some receivers have LCD displays and mapping software features, others simply provide time-stamped position values via a USB or Bluetooth connection. The position accuracy will also vary depending on the manufacturer. High end GPS receivers use both L1 and L2 signals and can make more accurate position calculations. Most common are the inexpensive receivers operating on just the L1 signals and require post processing to mitigate problems such as multipath error, noise and interference. The work presented in this project focuses on analyzing L1 data only.

Multipath Interference

As in any wireless communication system, microwave signals from GPS satellites will experience various types of noise and interference. Typical GPS signal noise and interference sources include atmospheric effects in the Troposphere and Ionosphere, timing errors and multipath interference errors. Multipath interference is predominant in highly developed urban areas. The phenomenon is caused as a result of two or more versions of the transmitted signal arriving at the receiver at slightly different times [15]. The different versions of the signal are combined at the receiver antenna to give a resultant

signal which can vary widely in amplitude and phase. These different versions of a signal are created during signal propagation as they reflect from various objects such as hills and buildings.

The multipath phenomenon is very difficult to model because the wireless medium which the signals travel through varies from one location to the next as well as in time. However, to obtain a basic understanding, it can modeled mathematically by first considering the direct line of sight (LOS) GPS signal defined as follows

$$S_d(t) = AP(t)\sin(\omega_c t)$$

where A is the amplitude of the signal, P(t) is the pseudo-random code, and ω_c is the carrier frequency of the transmitted signal. The multipath reflected signals can thus be modeled by the equation below

$$S_m(t) = \sum_{k=1}^{m} a_k AP(t-\delta_k)\sin(\omega_c t + \theta_k), \quad k = 1,..,m$$

where *m* represents the number of reflected signals, a_k, is the attenuation factor, δ_k is the delayed time, and θ_k is the phase shift caused by reflection of the signal from any physical object. Using the principle of superposition, the signal received at a GPS received will be the sum of these two equations as shown in the following equations.

$$S(t) = S_d(t) + S_m(t)$$

$$S(t) = AP(t)\sin(\omega_c t) + \sum_{k=1}^{m} a_k AP(t-\delta_k)\sin(\omega_c t + \theta_k)$$

In highly developed areas, the vast number of possible reflected signals seen at the receiver will all vary in amplitude and phase. As these are combined, the resultant signal will also vary in amplitude. This is known as fading. Considering the case or urban canyon areas, it can be assumed that no direct line of sight with the satellite is available, and the signal can be expressed as

$$S(t) = \sum_{k}^{m} a_k \cos(\omega_c t + \theta_k)$$

If the effects of motion are to be taken into account, additional parameters are required. It is necessary to define the parameter ζ_k which is the angle relative to the direction of motion of antenna at which the k^{th} reflected signal is received. The Doppler shift is thus given by

$$\omega_{d_k} = \frac{\omega_c v}{c} \cos(\zeta_k)$$

where v is the velocity of the receiver in motion, c is the speed of light, and ζ_k is uniformly distributed over $[0,2\pi]$. Considering motion, the multipath signal can be written as

$$S(t) = \sum_{k=1}^{m} a_k \cos(\omega_c t + \omega_{d_k} t + \theta_k)$$

Multipath fading is typically studied using probabilistic models to account for the changes in environmental characteristic in time. In the case of LOS, the Rician fading model is used. When no direct LOS can be assumed, the Rayleigh Fading statistical model is used, where the Rayleigh distribution is given by

$$p_r(r) = \begin{cases} \frac{r}{\sigma^2} e^{\left(\frac{-r^2}{2\sigma^2}\right)} & r \geq 0 \\ 0 & r < 0 \end{cases}$$

Current Approaches to Multipath Mitigation

Many multipath mitigation techniques have been developed and they are chosen for implementation depending on the application. A common technique is improving antenna gain pattern using choke rings. This is accomplished by eliminating multipath signals applying the principle of polarization. Another technique is to use a narrow correlator spacing and extend the multipath estimation delay lock loop. Signal and data processing such as exploring signal-to-noise ratio, using multiple reference stations, smoothing carrier phase and applying different filters are also methods to mitigate multipath. Traditional signal processing use denoising schemes based on Wiener filters. Recent research has moved focus towards filtering using non-linear methods. The use of wavelet analysis for

signal processing has been around for many years. Some previous work include that of Satirapod and Rizos [17], where wavelet analysis was applied on double differenced residuals to remove the multipath interference. Zhang and Bartone's [18] work on filtering using both wavelet and Wavesmooth techniques is another example. Investigations by Xia and Liu [19] on double differential phase observations also show the ability to mitigate multipath interference using wavelet analysis. In all these three studies, GPS receivers with dual frequency were used. This project applies wavelet analysis on only L1 frequency data. Other work also include interference excision and filtering based on local harmonic Fourier transform presented by R. Zarifeh [7].

Objective of the Project

The objective of this project is to study the Global Positioning System and learn how to apply wavelet analysis to mitigate the effects of multipath error on GNSS signals. In addition, the analysis is carried out using SystemC which allows flexibility to make changes and easily run simulations.

Organization of the Project

This project is organized into five chapters. Chapter 1 provides an introduction to Global Navigation Satellite Systems with focus on GPS and illustrates its use with a streetcar arrival schedule application. Chapter 2 provides information on the tools used during this project. It gives background information about system modeling using SystemC. This chapter also discusses wavelet analysis for GNSS signal filtering, GPS position estimation and the GPS toolkit. In Chapter 3, the SystemC implementation of the wavelet filter is described. Chapter 4 takes a look at how the GPS toolkit can help position estimation to verify the system. Finally, Chapter 5 provides a conclusion and some recommendations for future work.

Chapter 2

IMPLEMENTATION TOOLS

SystemC Modeling

In this project project, SystemC was used as the main implementation tool for the GNSS signal wavelet filter. SystemC is an open source library in C++ that provides a modeling platform for systems consisting of hardware and software components [20]. Similar to other Hardware Description Languages (HDL), SystemC allows users to construct structural designs using modules, ports and signals. While an implementation could well have be performed in other HDLs such as Verilog or VHDL, SystemC was chosen because it allowed development in a more familiar environment. Also, object oriented design inherited from C++, permitted the integration of the GPS toolkit for position estimation. In SystemC, modules can be instantiated within other modules, enabling structural design hierarchies to be built. Port and signals enable communication of data between modules, and all ports and signals are declared by the user to have a specific data type [11]. Some data types that are typically used include single bits, bit vectors, characters, integers, floating point numbers, vectors of integers, etc.

Concurrent behavior in SystemC is modeled using processes. Processes can be thought of as independent threads of control which assume execution when some set of events occur or some signals change, and then suspend execution after performing some action. In principle, processes are blocks of sequential code similar to functions. However, they differ from functions in that they are not called by the user. A simulation kernel contained in the SystemC class library is used to model the passing of time. This allows processes to be seen as always active and to be triggered by certain signals. There is a limited ability for specifying the condition under which a process resumes execution. That is, the process can

only be sensitive to changes to values of particular signals. Also, the set of signals to which the process is sensitive must be specified before simulation starts [2].

There are two main processes provided in SysmteC, namely, SC_THREAD and SC_METHOD. A thread by definition is a process that will automatically execute itself at the start of the simulation run and then suspend itself for the rest of the simulation. Threads can be halted at any moment and as often as necessary during execution. Although threads are meant to be executed only once throughout the simulation, typically it is required that they run for the whole duration of the simulation. This can be achieved by wrapping the function with an infinite loop. The SystemC simulation kernel knows to treat SC_THREAD processes as concurrent threads through process registration. Process registration is performed using the constructor of the module. The SC_METHOD process is similar to SC_THREAD but differ in that the former will run more than once and can not be suspended by a wait statement during execution. SC_METHODS will automatically go back to its beginning after it has finished executing.

A general purpose modeling foundation termed the *core language* enables system-level modeling such as Transaction-Level Modeling (TLM). TLM allows for the modeling, simulation and design of system architectures containing a combination of hardware and software components. The core language is then used to build elementary library models known as *elementary channels.*

The SystemC language architecture is shown in Figure 4. The diagram illustrates that everything is built on the C++ language and that upper layers are cleanly built on top of the lower layers. The core language is shown as providing only a minimal set of modeling constructs for structural description, concurrency, communication, and synchronization. Also, data types are separate from the core language and user-defined data types are fully supported. Typical communication mechanisms such as signals can be built on top of the

core language. Commonly used models of computation can also be built on top of the core language. It should be noted that lower layers can be used without the upper layers.

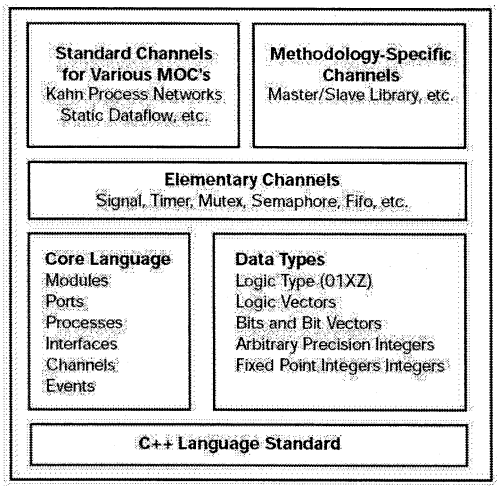

Figure 4: SystemC 2.0 Language Architecture

Wavelet Analysis of GNSS Signals

The tool used for signal processing in this project project is wavelet analysis. The fundamental idea behind wavelets analysis is to break up the signal into different frequencies and investigate each component in terms of scale. Wavelets are functions that satisfy certain mathematical requirements and are used to represent data or other functions [3]. The idea is similar to that of Fourier analysis which uses the sine and cosine functions to represent other functions. The advantage of using wavelets is that wavelets can adapt to the width of its time-slice according to the frequency components being extracted. The result is high resolution for high frequency components and low resolution for low frequency components. Further, Fourier transform assumes signals to be stationary. Since noisy GPS signals are non-stationary and vary with time, Fourier analysis shows limitations. The ability of wavelet analysis to zoom in at different levels makes it useful for processing signals whose spectral characteristics change with time [17].

Wavelet analysis is achieved by applying the wavelet transform to the signal of interest. The wavelet transform breaks up the signal into its "wavelets", which are scaled and shifted versions of the "mother wavelet". The two basic criteria for a function $\psi(t)$ to be considered a wavelet function are (1) that the average value of the wavelet must be zero and (2) that it must have unit energy. The criteria can also be written mathematically as follows:

$$\int_{-\infty}^{\infty} \psi(t)dt = 0$$

$$\int_{-\infty}^{\infty} \psi^2(t)dt = 1$$

Many mother wavelets have been identified and the choice of wavelet will vary depending on the application. In contrast to sinusoids, which are smooth and predictable, wavelets are typically irregular and asymmetric. A couple of mother wavelet examples include Meyer and Daubechies as shown in Figure 5.

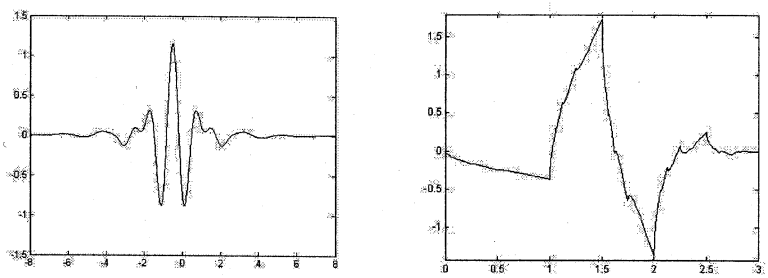

Figure 5: left: Meyer wavelet, right: Daubechies db2 wavelet

The concept of analysis in terms of scale can be better understood by comparing to the Fourier transform. The Fourier transform maps the signal into frequency components which lose time localization information. Windowing can be used to achieve localization but it is limited to one window size. This means that time resolution is the same for all frequencies components. On the other hand, the wavelet adapts the width of its time-slice

according to the frequency components being extracted [12]. The time scales are illustrated using time-frequency tiles such as the one in Figure 6.

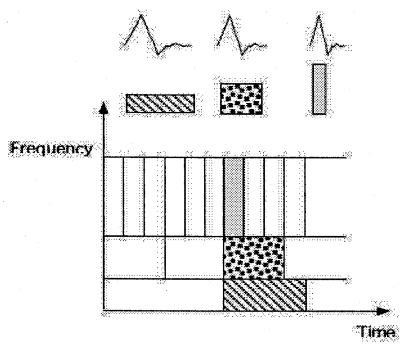

Figure 6: Time-frequency tiles with Daubechies basis function

Wavelet analysis can be classified in two types: The continuous wavelet transform and the discrete wavelet transform. The continuous wavelet transform can be thought of as the inner product between the original signal and scaled, shifted versions of the basis wavelet function. For the original signal $x(t)$ and the wavelet function $\psi(t)$ the continuous wavelet transform, CWT, is thus defined as:

$$CWT(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} x(t)\psi\left(\frac{t-b}{a}\right)dt$$

Where a represents the scale (dilation) parameter of the wavelet, b is the time-shift (translation) parameter of the wavelet, and t is time.

The discrete wavelet transform, DWT, is simply a sampled version of the continuous wavelet transform. In order to avoid redundant information which consumes unnecessary time and memory, the DWT can be implemented as a filter bank. The scaling and shifting parameters take on discrete values so that the wavelet coefficients can be described by two

integers, m and n. For the discrete signal $x[k]$ and wavelet function $\psi[n]$, the DWT is defined as:

$$DWT(m,n) = \frac{1}{\sqrt{a_0^m}} \sum_k x[k]\psi[a_0^{-m} n - k]$$

The DWT filtering process effectively separates the low and high frequency components. In terms of resolution, the low frequency components will have a large scale and the high frequency components will have smaller scale. The low frequency components typically contain most of the important information and are sometimes referred to as approximations. It is the low frequency components with coarse resolution that contain the main features of the original signal. The high frequency components with fine resolution are usually called details. It is in the detail coefficients where noise is typically found. This is analogous to removing the high frequency components from a voice signal. The voice may sound different but one can still understand what is being said. However, if the low frequency components are removed, what is left of the signal is typically just noise. DWT has also been found to be more efficient than the Fourier Transform. While the computational complexity required for a fast Fourier transform is $O(n\log_2(n))$, the DWT computational complexity is O(n).[16]

Multiresolutional analysis can be achieved by iteratively applying DWT on the computed approximations [1]. In other words, the wavelet transform is used to form a series of half band filters that divide a spectrum into high and low frequency bands. Multiresolutional analysis thus builds a pyramidal structure that is made up of successive wavelet transformations. The signal band is reduced by half on every step in the pyramidal structure. After the first pass, the signal band is split into two with one set of approximation coefficients (a1) and one set of detail coefficients (d1). The approximation coefficients (a1) is then passed through the filter a second time to obtain a new set of approximation coefficients (a2) and a new set of detail coefficients (d2). This process can be repeated as

required in order to achieve optimal results. This level of filtering will vary and depend on the nature of the signal. The mutiresolutional decomposition analysis is shown in Figure 7.

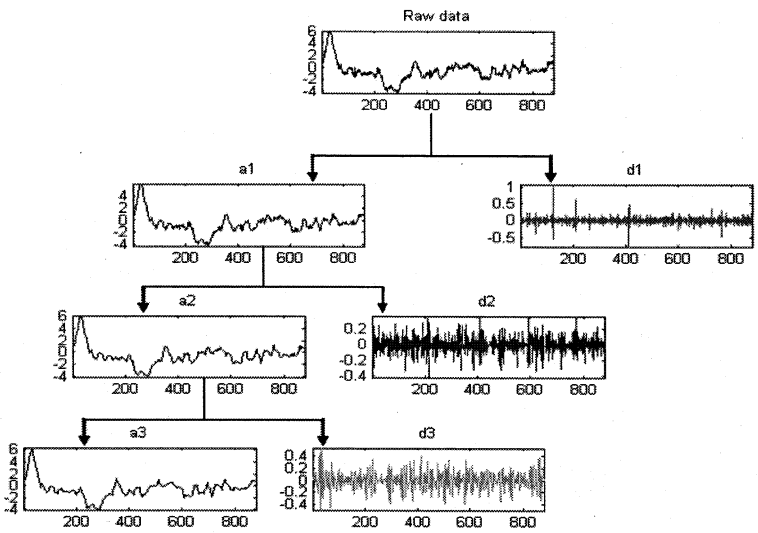

Figure 7: Three-level wavelet decomposition

Signal de-noising is attained by applying a method known as thresholding. The idea of thresholding is to manipulate the high frequency wavelet coefficients so that those under a certain threshold value are replaced by zeros. The de-noised signal is then obtained by the inverse transform of the modified coefficients. This technique is known as hard thresholding. Another technique called soft thresholding exists, in which coefficients above the threshold level are also modified by subtracting the amount of threshold. The threshold value can be estimated using different methods which evaluate parameters such as the signal standard deviation. A known method is Donoho's estimator which defines the amount of threshold in terms of a logarithmic function of the signal length and standard deviation. The amount of thresholding can also be adjusted depending on the nature of the signal properties, effectively creating a type of adaptive thresholding. In this work, hard

thresholding de-noising is used on wavelet coefficients from pseudorange signals obtained from GPS satellites.

GPS Position Estimation

To determine position at least four satellite time signals are required. The time signals are used to calculate the signal travel times $\Delta t_1 \ldots \Delta t_4$. These travel times are then used to calculate the pseudoranges $R_1 \ldots R_4$. As the locations X_{Sat}, Y_{Sat} and Z_{Sat} of the four satellites are known, the user's position can be calculated. A Cartesian, three-dimensional coordinate system with a geocentric origin, as shown in Figure 8, is used initially [6].

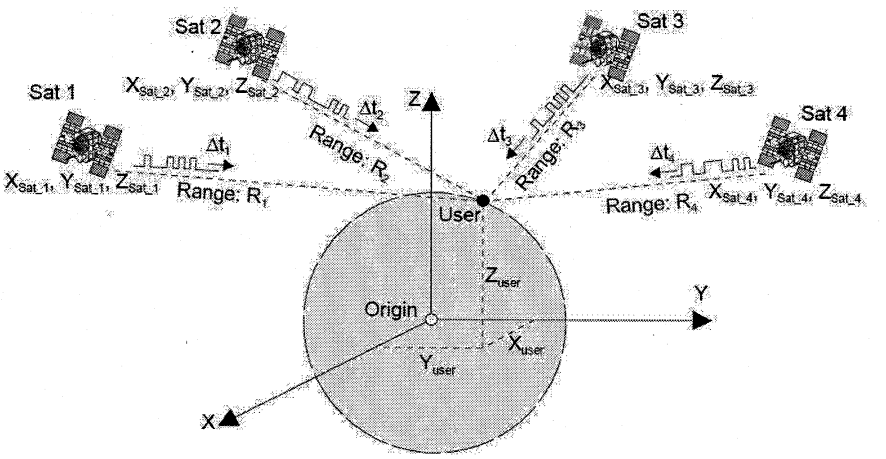

Figure 8: Three-dimensional coordinate system

Due to receiver clock inaccuracies and synchronization issues, the measured travel times will have an error, Δt_0. To compensate, the measured pseudorange value, PSR, will vary from the true range, R, by a factor of $c \cdot \Delta t_0$. Therefore, for satellite i, the pseudorange calculation can be expressed as

$$PSR_i = \sqrt{(X_{Sat\_i} - X_{User})^2 + (Y_{Sat\_i} - Y_{User})^2 + (Z_{Sat\_i} - Z_{User})^2} + c \cdot \Delta t_0$$

With four satellites, a set of four non-linear equations are formed. In order to solve them, a Taylor series expansion is applied to the root function. The estimated position will have an error due to the unknown variables Δx, Δy and Δz as shown by the following equations:

$$X_{User} = X_{Total} + \Delta x$$
$$Y_{User} = Y_{Total} + \Delta y$$
$$Z_{User} = Z_{Total} + \Delta z$$

The distance R_{Total} from each satellite to the estimated position is calculated in a similar fashion and can then be used together with the Taylor series expansion to give the following equation

$$PSR_i = T_{Total\_i} + \frac{\partial(R_{Total\_i})}{\partial x} \cdot \Delta x + \frac{\partial(R_{Total\_i})}{\partial y} \cdot \Delta y + \frac{\partial(R_{Total\_i})}{\partial z} \cdot \Delta z + c \cdot \Delta t_0$$

After evaluating the partial differentiation, the equation becomes

$$PSR_i = T_{Total\_i} + \frac{X_{Total} - X_{Sat\_i}}{R_{Total\_i}} \cdot \Delta x + \frac{Y_{Total} - Y_{Sat\_i}}{Y_{Total\_i}} \cdot \Delta y + \frac{Z_{Total} - Z_{Sat\_i}}{Z_{Total\_i}} \cdot \Delta z + c \cdot \Delta t_0$$

The equations can now be written in matrix form as shown below and the position delta values Δx, Δy and Δz can be solved for using linear algebra.

$$\begin{bmatrix} PSR_i - T_{Total\_1} \\ PSR_i - T_{Total\_2} \\ PSR_i - T_{Total\_3} \\ PSR_i - T_{Total\_4} \end{bmatrix} = \begin{bmatrix} \frac{X_{Total} - X_{Sat\_1}}{R_{Total\_1}} & \frac{Y_{Total} - Y_{Sat\_1}}{Y_{Total\_1}} & \frac{Z_{Total} - Z_{Sat\_1}}{Z_{Total\_1}} \\ \frac{X_{Total} - X_{Sat\_2}}{R_{Total\_2}} & \frac{Y_{Total} - Y_{Sat\_2}}{Y_{Total\_2}} & \frac{Z_{Total} - Z_{Sat\_2}}{Z_{Total\_2}} \\ \frac{X_{Total} - X_{Sat\_3}}{R_{Total\_3}} & \frac{Y_{Total} - Y_{Sat\_3}}{Y_{Total\_3}} & \frac{Z_{Total} - Z_{Sat\_3}}{Z_{Total\_3}} \\ \frac{X_{Total} - X_{Sat\_4}}{R_{Total\_4}} & \frac{Y_{Total} - Y_{Sat\_4}}{Y_{Total\_4}} & \frac{Z_{Total} - Z_{Sat\_4}}{Z_{Total\_4}} \end{bmatrix} \cdot \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta t_0 \end{bmatrix}$$

The solution of Δx, Δy and Δz is then used to re-calculate new position estimates.

$$X_{Total\_New} = X_{Total\_Old} + \Delta x$$

$$Y_{Total\_New} = Y_{Total\_Old} + \Delta y$$

$$Z_{Total\_New} = Z_{Total\_Old} + \Delta z$$

The process is repeated using the new estimated values X_{Total_New}, Y_{Total_New}, Z_{Total_New} until the position delta values are smaller than the desired error.

GPS Toolkit

The GPS Toolkit, also referred to as the GPStk, is open source software developed by the Applied Research Laboratories at the University of Texas at Austin [8]. The GPStk provides a core library and collection of applications that support GPS research, analysis and development. The software is written in C++ and makes use of object-oriented programming principles to ensure modularity, extensibility and maintainability. It supports a broad range of functionality [9].

1. *RINEX input and output.* The Receiver Independent Exchange (RINEX) format is the most widely used standard format definition for the storage of GPS observations, GPS navigation message information and meteorological data associated with GPS observations. The GPStk is able to read, write and process data from RINEX files

2. *Ephemeris evaluation.* The library provides complete satellite ephemeris storage capabilities. The ephemeris storage classes effectively encapsulate the details of ephemeris handling.

3. *Date and time conversions.* The "DayTime" class can manipulate various time formats such as modified Julian date, GPS time and calendar dates.

4. *Matrix and vector algorithms.* An extensive matrix and vector package includes the usual arithmetic operators like addition, subtraction, multiplication and division. LU decomposition, singular value decompo-sition, Cholesky decomposition and inversion of matrices are available.

5. *Mathematical and statistical algorithms.* These include polynomial fitting and Lagrange interpolation.

6. *Tropospheric and ionospheric modeling.* Includes an ionospheric model using least squares and the slant TEC values.

7. *Position determination.* Position estimation with various error atmospheric error correction is supported.

With RINEX files available from various observations station around the world, the GPStk can be used as a source of GPS raw data that can be used for further processing. The pseudoranges from a RINEX file can be extracted and fed into the wavelet filter module for de-noising. The GPStk position determination functionality is able to further process the output from the filter to finally give a set of more precise position values.

Chapter 3

WAVELET FILTER IMPLEMENTATION

GNSS signal processing using a wavelet based filter is performed all within one hardware module. The module makes use of abstract software components in order to carry out some of the mathematical operations. The module accepts pseudorange information on one of the data ports. The module reads in these values and performs the Discrete Wavelet Transform (DWT) using the Daubechies db4 mother wavelet. De-noising with hard thresholding is applied to high frequency wavelet coefficients. An inverse discrete wavelet transform (IDWT) then reconstructs the pseudorange signal. The new pseudorange values are provided on the output data port of the module. This design allows for the filtering mechanism to run independently and gives flexibility to test different input sources. The filtering algorithm was tested and evaluated using two different sources. The first pseudorange source examined was an auto-generated known signal with added Gaussian noise. The second source was real pseudorange data from an observation station in Victoria, British Columbia extracted from a RINEX file provided by Natural Resources Canada. Output from the filter can be written to a file for analysis or used to perform further computation such as position estimation. The system thus consists of three main components: source, filter and sink, as shown in the figure below.

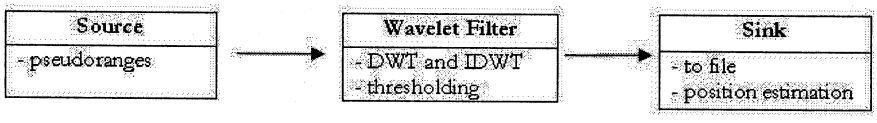

Figure 9: GPS filter system block diagram

Two types of signals are passed between the three modules, namely, control signals and data signals. The data signals are the ones containing the information about the pseudorange ("PR in" and "PR out") and in the second implementa-tion, the satellite number ("Sat in" and "Sat out"). The control signals are used as flags or semaphores, effectively communicating between modules when data is required, ready or received. In addition, a clock signal is required on each of the modules to drive each component. Data and control signals are discussed in more detailed in the following section. A detailed block diagram with corresponding signals can be seen in Figure 10.

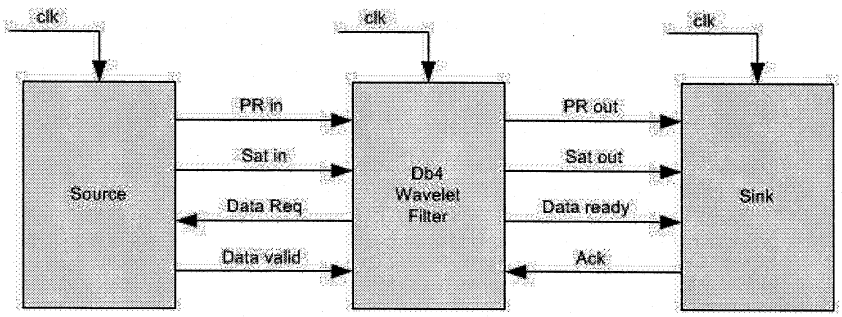

Figure 10: System block diagram with signals

Wavelet Filter Module

The wavelet filter module applies the wavelet analysis and de-noising techniques discussed in chapter 2. The DWT and IDWT are implemented in software as filter banks processing pseudorange values read from the input line 20 samples at a time. The module uses one main process called "dwtfilter" implemented as a SystemC SC_THREAD which runs in an infinite loop. The thread requests for pseudorange data and when data is available it processes it. The data is first stored in temporary memory and then used to call the DWT and IDWT software functions.

The module is basically a C++ struct defined as a SystemC sc_module. It uses four data ports and four control ports. The four data ports used in this module are used to pass

pseudorange and satellite number data in and out. The pseudorange data is used directly by the DWT and IDWT functions. The satellite number is used to determine how to group data before it can be filtered. It also provides information on the output ports as to which satellite the pseudorange belongs to.

The control ports allow control signals to be used as semaphores between all three modules and are sufficient for processor timing without the need of a complex scheduler design. Two control ports are used to interface with the source module and the other two with the sink module. The "data_req" port is an output port and to which the filter module needs to set a signal when it is ready to accept the next pseudorange value. The source module is responsible for providing the next pseudorange value and sending a response signal to the "data_valid" port, to communicate to the filter module that the data is ready. The filter module clears the signal on the "data_req" port, reads in the value and sets the "data_req" port again when ready. Similarly, after the filter computation is complete, the filter module will put the new pseudorange value on the data output port and set a signal on the "data_out_ready" port to notify the sink module. The sink module is responsible for acknowledging that the data has been received by responding to the "data_out_ack" port. All signaling and data processing is controlled within the "dwtfilter" thread which is set sensitive to execute on the positive edge of the clock signal.

Source Module

The source module is responsible for providing GPS pseudorange data to the wavelet filter module. Two source modules were developed. The first source module made use of a text file containing the pseudorange values. The file can be thought of as containing a vector of pseudorange data. When data is requested by the filter module, the source module will read the text file and get the next pseudorange value and provide it to the filter module. The second source module makes use of the GPStk to read real pseudorange values from a RINEX file and provide the to the filter module when requested. This will be discussed further in the next chapter.

The text file source module uses two data ports and two control ports. One data port is used to provide pseudorange information. The second data port is set to a constant value to simulate data from only one satellite. The first control port listens for the "data req" signal from the filter module. When data is requested, the pseudorange information is placed on the "out_pr" data port and a signal is set on the "data_valid" port.

The source module runs one main process called "entry" implemented as a SystemC SC_THREAD whose execution is sensitive to the positive clock edge. All signaling and data manipulation is performed by this process

Sink Module

The sink module receives the output from the filter module and provides a means to interpret the filter results. Two sink modules were developed to match the two source modules. The first one basically receives new pseudorange values from the filter module and writes them back on to a text file as a vector of pseudoranges. The second sink module uses the new pseudoranges to calculate the position values. This is explained in more details in the next chapter.

The text file sink module uses two data ports and two control ports. One data port is used to read pseudorange information. The second data port reads the constant satellite value which is discarded. The "data_in_ready" port listens for the filter module to send a signal when new filtered data is ready. When data is ready, the pseudorange information is read by the sink module. It then provides a response on the "data_in_ack" control port to notify the filter module that the value was successfully received.

Similar to the source module, the sink module also runs one main process called "entry" implemented as a SystemC SC_THREAD whose execution is sensitive to the positive clock edge. All signaling and data manipulation is performed within this thread.

System Integration and Analysis

The three modules are linked together in the main program from within the top level file. The main program defines all required signals including the system clock (set at 10 ns period and 50% duty cycle) and instantiates all three objects. Following the instantiation of each module, port binding is performed to effectively wire the modules together.

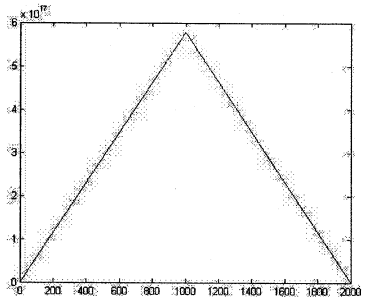

Figure: 11: Cross-correlation between original and filtered signals

A simulation of the system was run using the text file based source and sink module and a known pseudorange signal with added random noise. Note that random noise only differs from multipath interference in that random noise can be thought of as a stationary process where as multipath can have non-stationary characteristics. However, for the purpose of initial demonstration, random noise should be sufficient. The original pseudorange values and the new filtered values were analyzed using the cross-correlation and correlation coefficients functions in Matlab. The correlation coefficients obtained were [1.0000, 0.9975; 0.9975, 1.0000] and the cross-correlation function plot is shown in Figure 11. These results show that good statistical correlation exists between the original pseudorange signal and the new filtered signal.

The modules can also be analyzed in terms of how the data is passed through the system. The filter module first requests 20 pseudorange sample values before it starts computation. After the values are received the computation begins and the filtered values can be seen on

the output port. The actual system signaling and timing can be analyzed by creating a vcd trace file from in the SystemC top level program. The vcd file can be translated into a wlf file in order to be viewed in ModelSim [21]. The simulation results showed sequential operation between the first two modules. In other words, the source module provided the first set of data and then stopped running until the filter module was ready to receive data again. The filter module needs to complete computation of the first block before it can request the next batch of data. The sink module process effectively runs concurrently with the other processes as it is able to execute all its tasks at the same time as the filter module. The sink module thus runs somewhat independently of the filter and source modules. The observed system signals are shown in Figure 12.

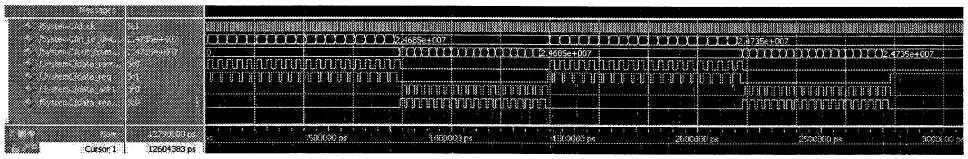

Figure 12: System signals between the three modules

The top signal is the clock signal running at 100 MHz. The second signal from the top is the pseudoranges entering the filter module. The third signal is the filtered data coming out from the filter module. The fourth signal is the "data valid" signal from the source module. The fifth signal is the "data required" signal from the filter module. Note that these last two signals are offset with different duty cycles but share the same frequency as data is exchanged between the two modules. The last two active signals are the "acknowledge" and "data ready" signals used by the sink.

Chapter 4

POSITION ESTIMATION IMPLEMENTATION

The previous chapters have already described the overall system consisting of the source, sink and filter modules and how they interact. In this chapter, position estimation and the use of the GPStk is discussed. The alternative source and sink implementations are presented and the wavelet filter module performance is evaluated. Note that for this evaluation, the wavelet filter module sample size was increased to 120 samples and the level of filtering was increased from 1 to 3. The new source module uses the GPStk to read real pseudorange values from a RINEX file. The analysis in this document uses data from an observation station in Victoria, British Columbia extracted from a RINEX file provided by Natural Resources Canada [22]. Since the observation station is not located in the center of a highly developed urban area, the observed pseudorange values do not show much distortion. Rinex data for an observation station experiencing multipath was not readily available, thus random noise is added to the pseudorange values for analysis. The data is passed through the wavelet filter and the new sink module performs further computation. The sink module first generates a new RINEX file with the new pseudorange values and then performs the position estimation calculation. The position values are written to a Matlab m file effectively creating two vectors representing the x and y position coordinates.

New Source Module

The modified source module maintains the same hardware structure as the first implementation, as no ports or processes are removed or added. Working in SystemC, the changes can actually be modeled in software using the GPStk. The GPStk namespace is added to the source which allows RINEX files to be read. In order to do this, two more object types are required, namely, RinexObsData and RinexObsStream. A RinexObsStream

object is instantiated with the input file name which allows a RINEX file to be opened for reading. The "entry" thread running in the source module loops through the data epochs for all satellites and records the values in a RinexObsData object. The pseudorange information is then extracted from the observation data object and passed on to the source module pseudorange data port. Note that only L1 information is used. Control signals between the filter module and the source module are exchanged in the same way as described in the previous section.

New Sink Module

Similar to the new source module, the modified sink module also does not add or remove hardware ports or processes. The changes and additions are modeled in software by adding the GPStk namespace. The new module uses three GPStk objects for RINEX file manipulation, namely, RinexObsHeader, RinexObsData and RinexObsStream. Two RinexObsStream objects are instantiated, one to read the original RINEX file and the other to create the new modified RINEX file. The RinexObsHeader object is used to copy the header information into the new file. The sink module "entry" thread, loops through the data epochs and for each satellite, it populates the RinexObsData object with a combination of previous observation information and new buffered filtered data. Only information pertaining to L1 values is modified. After looping though all data epochs, the observation data object is used to finish populating the new RINEX file. The filtered data is obtained from the filter module and stored in the buffer using the same control signals mechanism described in the previous section.

After the new RINEX file is generated, the sink module calls a new function named getPositions. This function uses more GPStk objects for handling navigation information, meteorological data, tropospheric model, position calculation using the Bancroft algorithm, storing ephemeris information, and defining some GPS system constants. The function requires as input a RINEX observation file and a RINEX navigation file. A third RINEX meteorological file is optional and is not used in this analysis. The tropospheric modeling is

skipped when no meteorological data is available. The RINEX navigation file is read in the same way as the observation file and the information is stored in navigation data and navigation header objects. Similarly, the ephemeris information is stored in temporary memory. The observation file header is stored in an observation header object and the pseudoranges are read one epoch at a time. For each epoch, a satellite map object is used to populate a matrix of satellite positions and corresponding pseudorange values. The matrix populated in each epoch is then used by the Bancroft function to calculate the x, y, z position values. The position values are expressed in meters for an Earth Centered, Earth Fixed (ECEF) reference frame. The position solution values for x and y are appended to a Matlab file that generates two vectors corresponding to each one of the two directions. The z direction is omitted for ease of plot interpretation.

The Bancroft algorithm applied in this implementation uses vector and matrix operations to efficiently calculate the GPS position [13]. The algorithm uses the generalized inverse $(H^T W H)^{-1} H^T W$ where H is a measurement matrix and W is positive definite weighting matrix [14]. To expand on the algorithm, the pseudorange value for a given satellite can be expressed as

$$t_i = d(\mathbf{x}, \mathbf{s}_i) + b, \quad 1 \le i \le n$$

where $\boldsymbol{x}$, is the user ECEF position coordinates, $\boldsymbol{s}_i$ is the ith satellite's ECEF position coordinates, $d(\boldsymbol{x},\boldsymbol{y})$ is the distance from $\boldsymbol{x}$ to $\boldsymbol{y}$ and b is the clock offset. A 1x4 column data vector $\boldsymbol{a}$ is defined for each satellite

$$\mathbf{a}_i = (\mathbf{s}_i^T t_i)^T, \quad 1 \le i \le n$$

where T denotes the transpose. The Minkowski functional for 4-space is defined by

$$\langle \mathbf{a}, \mathbf{b} \rangle = a_1 b_1 + a_2 b_2 + a_3 b_3 - a_4 b_4$$

Parameters $\mathbf{A}$, i_o, r_i and $\mathbf{r}$ are defined as follows

$$\boldsymbol{A} = (\mathbf{a}_1\ \mathbf{a}_2\ \mathbf{a}_3 \ldots \mathbf{a}_n)^T$$

$$\boldsymbol{i}_0 = (1\ 1\ 1 \ldots 1)^T$$

$$r_i = \langle \mathbf{a}_i, \mathbf{a}_i \rangle / 2$$

$$\mathbf{r} = (r_1\ r_2\ r_3\ \ldots\ r_n)^T$$

The generalized inverse is then computed as shown below

$$\boldsymbol{B} = (\boldsymbol{A}^T \boldsymbol{W} \boldsymbol{A})^{-1} \boldsymbol{A}^T \boldsymbol{W}$$

where $\boldsymbol{W}$ is a symmetric positive definite weighting matrix such as the identity matrix. Using the generalized inversed, the 1x4 vectors **u** and **v** are computed

$$\mathbf{u} = \boldsymbol{B}\boldsymbol{i}_0$$

$$\mathbf{v} = \boldsymbol{B}\boldsymbol{r}$$

The scalar coefficients **E, F, G** are defined using vectors **u** and **v** as follows

$$\mathbf{E} = \langle \mathbf{u}, \mathbf{u} \rangle$$

$$\mathbf{F} = \langle \mathbf{u}, \mathbf{v} \rangle - 1$$

$$\mathbf{G} = \langle \mathbf{v}, \mathbf{v} \rangle$$

The scalar coefficients are written in a quadratic and solved for a pair of roots $\lambda_{1,2}$

$$\mathbf{E}\lambda^2 + 2\mathbf{F}\lambda + \mathbf{G} = 0$$

The 1x4 column vectors $\boldsymbol{y}_{1,2}$ can be computed as per the following definition

$$\boldsymbol{y}_{1,2} = \lambda_{1,2}\mathbf{u} + \mathbf{v}$$

Solving for either pair $\boldsymbol{x}_1 b_1$ or $\boldsymbol{x}_2 b_2$ in the following equation will provide the solution for user position and clock offset.

$$\boldsymbol{y}^T = (\boldsymbol{x}^T - b)^T$$

System Evaluation

In order to verify the system, the source Rinex file needs to contain some noise to be filtered. The Rinex files obtained from Natural Resources Canada contained data from GPS receivers positioned in open areas away from urban development and therefore did not contain much multipath distortion. Rinex information for GPS receivers positioned in a downtown environment was not available. To compensate, Matlab was used to add random noise to the available Rinex files. This was achieved by extracting the L1 values from the Rinex file, importing these into Matlab where noise was added and written back to another

file. This file was then used to modify the L1 values in the Rinex file. Two data sets were created and used to evaluate the performance.

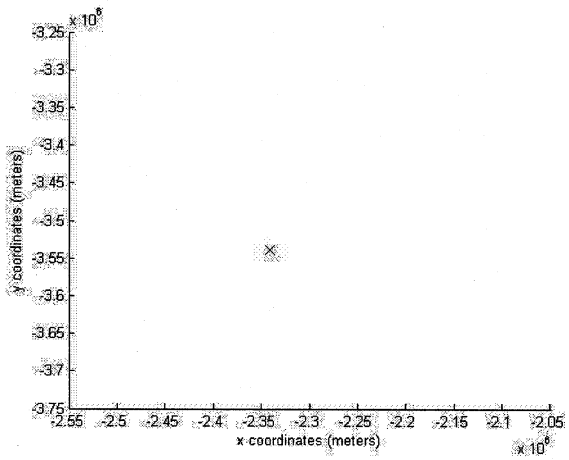

Figure 13: Ideal position plot

Under ideal conditions, the position values will converge to a single point. For the purposes of illustration, the x and y coordinates points of an ECEF reference frame are used. The observation station in Victoria, British Columbia, has ECEF coordinates (-2341332.134580, -3539049.273172, 4745791.567740) with equivalent geodic latitude and longitude coordinates (48.38978335, -123.4874742). The ideal case can thus be plotted as shown in Figure 13.

Performance can measured in terms of amount of visual clustering between the x and y points of ECEF position values. The figure of merit used is the percentage of reduced area that the position points fill up. The area that the points fill up is calculated by adding up unit square areas where the data points exist. The unit square area is the area created when a grid is drawn on the plot. The percentage of reduced area is thus calculated by taking the ratio of reduced cluster size to original cluster size. Both data sets used showed improvement in position accuracy as can be seen in the following figures. The first data set

was created by adding random noise to a low precision set of pseudorange values. The x and y values of the noisy signal were plotted and are shown in Figure 14 below. The red x in all plots represents the actual position.

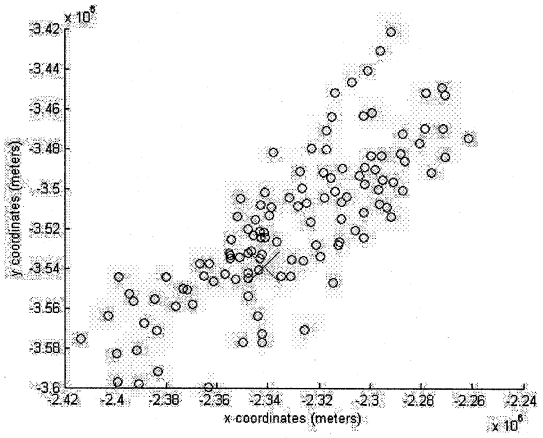

Figure 14: Data set 1 before wavelet filtering

After passing through the wavelet filter with hard thresholding set at 500,000, the cluster size reduced by 1-(24/35)*100 = 31% as can be seen in Figure 15.

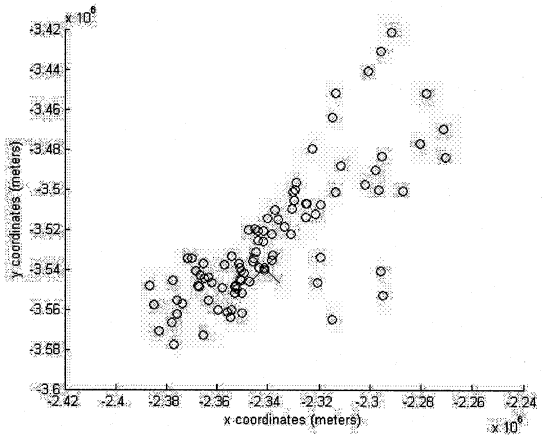

Figure 15: Data set 1 after wavelet filtering

The second data set was created by adding random noise to a high precision set of pseudorange values resulting in a noisier signal as shown in Figure 16.

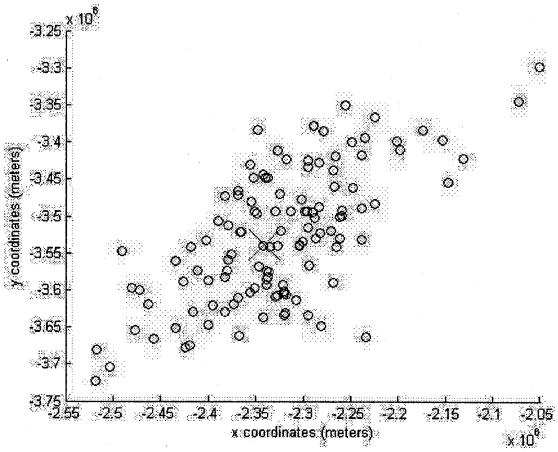

Figure 16: Data set 2 before wavelet filtering

When passed through the wavelet filter, the position values start to converge toward the actual position point (-2341332, -3539049). As can be seen in Figure 17, the cluster size is reduced by 1-(21/35)*100 = 43%.

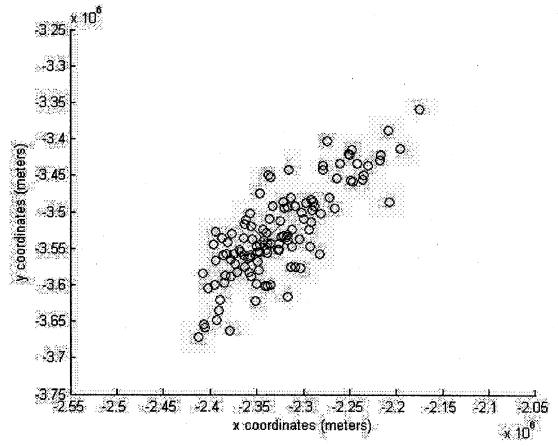

Figure 17: Data set 2 after wavelet filtering

It should be noted here that while moderate improvement in position estimation accuracy is achieved using these two data sets, better performance is expected by using data sets experiencing actual multipath distortion [5]. Different values of thresholding were analyzed and it was found that a significantly high value was required to achieve valuable results. In both examples above, the threshold value was set to 500,000, clearing the majority of the detail coefficients. The threshold value was chosen by various trial attempts starting out with an estimate value using Donoho's estimator. For a threshold *th*, standard deviation σ, and signal of length n, the estimator is given by the following equation

$$th = \frac{\sigma}{\sqrt{n}}\sqrt{2\log(n)}$$

However, with the larger coefficients being in the range of $6x10^7$, the values being cleared were 2 orders of magnitude smaller. The thresholding technique used was a simple hard threshold method. Soft and adaptive thresholding may be used and are expected to further improve position estimation accuracy and reduce cluster size. However, the gains should come at a speed performance cost due to the added computational complexity required.

Looking at the system signals for the position estimation implementation, it can be seen that the pattern is the same as in Figure 12 of chapter 3. Data is passed between Source and Filter modules for 4.8μs, before the filter starts computation and begins making the filtered values available on the output port. This can be seen in Figure 18. The difference now is that Source-Filter communication time for a given sample set is longer because the number of samples was increased from 20 to 120. This is also true for Filter-Sink communication. This has no negative effect on the filtering algorithm but rather provides more samples to be used at different levels of filtering.

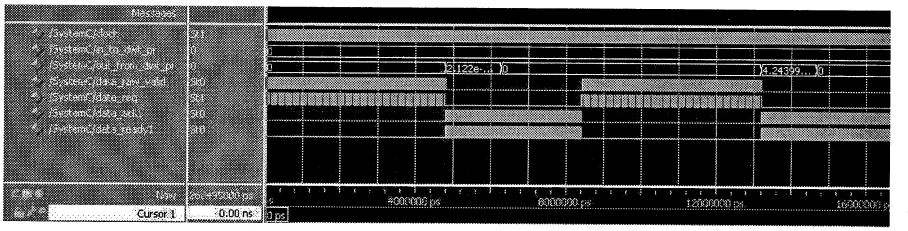

Figure 18: System signals for position estimation implementation

Taking a closer look at the system signals, the overhead introduced by inter-module communication can be measured. Taking the time difference between two successive data requests from the filter module, the time interval to request and transfer data from the source module is calculated. This interval is equivalent to the period of both the filter module "data required" and the source module "data valid" signals. Figure 19 bellow shows this value to be 4 clock cycles. With the 10 ns clock, this results in 40 ns. Note here that it takes 2 clock cycles for the data from the source module to become ready.

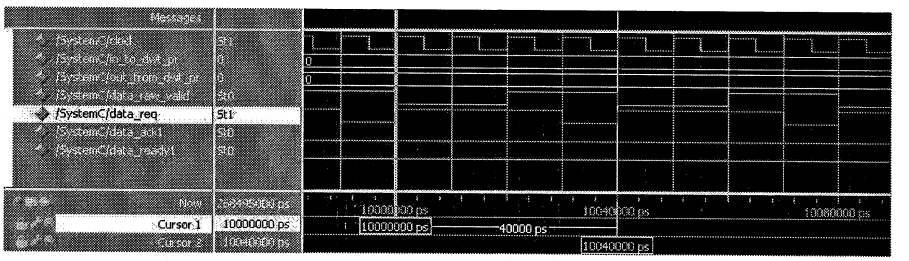

Figure 19: System signals during Source-Filter communication

Chapter 5

CONCLUSIONS AND RECOMMENDATIONS

In this work, the GPS system and its signals were studied from a signal processing perspective. Focus was concentrated on filtering noise in GPS pseudorange data. The implementation could potentially then be used to mitigate multipath distortion found in highly developed urban centers such as the city of Toronto. This project illustrates that mitigating these effects in real receivers for real applications is a task that can be accomplished. Thus, applications such automobile navigation and streetcar scheduling can be made more accurate and reliable, providing a better end user experience. This is demonstrated by developing a filtering system implementation in SystemC language with the aid of the GPS Toolkit developed by the University of Texas Austin. SystemC architecture and basic concepts were studied and are explained here. Wavelet theory was discussed to introduce the wavelet based filter used to mitigate the effects of multipath in GPS signals. Multiresolutional analysis was investigated and applied in the Daubechies wavelet filter. Position estimation was studied. The implementation developed in this work used the Bancroft algorithm to estimate initial position values. Position plots obtained from the simulations clearly showed an improvement in the position estimation accuracy by the examining the percentage reduction in cluster size.

Noting that position plots used are generated by plotting x and y ECEF coordinate values, the actual latitude and longitude values can also be calculated and drawn on a world map. GPStk provides coordinate conversion making this task fairly simple.

The results showed moderate improvement in position estimation after passing through the filter module. This study can be continued to verify that cluster size can be reduced

even more. A first recommendation would be to obtain real GPS data from within a downtown environment. The data can be stored in Rinex format and fed to the wavelet filter. Different software to convert GPS data from different vendors to Rinex format is available. With data in Rinex format, the data can easily and effectively be examined using the existing implementation.

Alternatively, one could take a step back and re-evaluate the SystemC implementation. With current embedded systems running on flexible operating systems such as ucLinux, a complete software implementation can be developed. The work done here, can serve as a good starting point. SystemC modules, data and control signals can be replaced by a single program reading values from memory. Data collected from a GPS can still be stored to Rinex format or read directly for more real-time like performance. Reading data directly from a GPS receiver can feed the filter module with more samples per time interval which can help adjusting the level of filtering.

The de-noising mechanism can also be an area for further research. In the experiments carried out for this project, only hard thresholding was used. The work by M. Aram [1] shows that different techniques including soft and adaptive thresholding can be applied for better position estimation. Threshold values based on eigenvalues were proven to be more effective in GPS applications and it would be interesting to see implemented as an addition to this work.

Another recommendation is to add flexibility to the filter module and extend the use of just a Daubechies db4 wavelet to other wavelets. Extensive research would be required to examine different signals sources against different mother wavelet in an attempt to find a relationship between them. If a relationship can be found an adaptive wavelet filtering algorithm can be developed. Changes in signal properties can be monitored to adjust the mother wavelet used in the filter module to achieve better position estimation.

In the first implementation presented in this project, data was filtered and examined based on the cross-correlation results between the original and filtered signals. The wavelet filter module only performed a one level wavelet decomposition analysis. In the second simulation, consisting of actual GPS position points with added noise, three levels of filtering were used. For signals with lower signal to noise ratios, increasing the number of levels helps improve performance. This can also be made an adaptive mechanism built on entropy based calculations.

Another recommendation for future work using the SystemC implementation is to try alternative inter-module communication mechanisms. The current control signals work well between few processes, if this is integrated with other systems or modules the system performance can start to degrade as the exchange of control signals becomes unwieldy. The use of a scheduler may be considered to better allocate resources for processing. Different scheduler mechanisms can be studied to determine the most appropriate.

In the advent of Galileo and GLONASS, wavelet filter simulation for these systems can also be performed. If these systems where to become available in the near future, information from these new satellites could also be used in a second experiment for the same location. It is expected that the added redundancy could help further improve the position determination accuracy.

Having completed thorough hardware simulation of the wavelet filter, future work in this area can consist of actual hardware implementation. Software for SystemC to VHDL and/or Verilog conversion is available. There may be limitations to this software which should be taken into account prior to being used. Also, it will be important to research an appropriate GPS receiver which can be tapped into for real-time feed of pseudorange values. Note that not all vendors provide sufficient documentation or support. Alternatively, with more effort, the whole GPS receiver

could be built by putting together an antenna, a processor and a clock. However, demodulation and additional signal processing not covered in this project may be required.

In conclusion, the wavelet analysis has proven to be of value for the denoising of GNSS signals. The wavelet based filter is able to improve position estimation accuracy and was shown through reduced position plot cluster size. The SystemC implementation was able to confirm that wavelet analysis can effectively be used in filtering noise in GPS signals. The implementation also provides a starting point for further work for a physical implementation of a wavelet based filter.

BIBLIOGRAPHY

[1] M. H. Aram, "Multiresolutional Characterization and Mitigation of GNSS Signal for Robust Positing" Thesis for the degree of Master of Applied Science, Ryerson University, Toronto, Ontario, Canada, 2007

[2] S. Swan, "An Introduction to System Level Modeling in SystemC 2.0" Open SystemC Initiative, 2001

[3] A. Graps, "An Introduction to Wavelets" IEEE Computational Sciences and Engineering, 1995

[4] K. Bjerge, "Guide for getting started with SystemC development", Danish Technological Institute, 2007

[5] M. Aram, B. Li, S. Krishnan, A. Anpalagan, "Improving Position Estimates from a Stationary GNSS Receiver Using Wavelets and Clustering", in Proceedings of IEEE Canadian Conference on Electrical and Computer Engineering, pp. 758-762, May 2006.

[6] J. Zogg, "Essentials of Satellite Navigation", u-blox Corporation, Zurcherstrasse, Switzerland, 2006

[7] R. Zarifeh, "Interference Exicsion and Filtering in Spread Spectrum Communiation Systems", University of Hertforshire, UK, 2007

[8] Applied Research Laboratory (ARL), at the University of Texas at Austin, GPS Toolkit website http://www.gpstk.org/bin/view/Documentation/ WebHome

[9] B. Tolman, R. Harris, T. Gaussiran, D. Munton, J. Little, R. Mach, S. Nelsen, B. Renfro, D. Schlossberg, "The GPS Toolkit – Open Source GPS Software" ARL, The University of Texas at Austin and School of Information and Management Systems, The University of California at Berkley, Long Beach California, ION-GNSS-2004

[10] Mathworks website http://www.mathworks.com/

[11] "IEEE Standard SystemC Language Reference Manual", IEEE Computer Society, IEEE Std 1666, 2005

[12] S. Mallat, "A Wavelet Tour of Signal Processing", Academic Press, 2nd Edition, September 1999

[13] R. Puri, A. Kaffas, "Multipath Mitigation of GPS Signals for an on-board Unit for Mobility Pricing" B. Eng. Thesis, Ryerson University, Toronto, 2005

[14] S. Bancroft, "An Algebraic Solution of the GPS Equations" IEEE Transactions on Aerospace and Electronic Systems, Vol. AES-21, No. 7, January 1985

[15] T. S. Rappaport, "Wireless Communications – Principles and Practices" Prentice Hall, Upper Saddle River, New Jersey, 1996

[16] B. Vidakovic, P. Mueller, "Wavelet, A Tutorial Introduction", Duke University, 1991

[17] C. Satirapod, C. Rizos, "Multipath Mitigation by Wavelet Analysis for GPS Base Station Applications", Survey Review, Vol. 38, No. 295, 2003

[18] Y. Zhang, C. Bartone, "Real-Time Multipath Mitigation with Wavesmooth Technique using Wavelets", ION GNSS 17th International Meeting of the Satellite Division, pp. 1181-1193, September 2004

[19] L. Xia, J. Liu, "Approach for Multipath Reduction Using Wavelet Algorthim", The Institute of Navigation (ION), International Technical Meeting, pp. 2134-2143, Salt Lake City, Utah, 2001

[20] G. Kahn, EE8205: Embedded Computer Systems Course notes and Website. http://www.ee.ryerson.ca/~courses/ee8205/ (last accessed on September 16, 2008)

[21] Mentor Graphics, "ModelSim Tutorial", Software Version 6.3e, Wilsonviller, Oregon, February 2008

[22] Natural Resources Canada, Canadian Spatial Reference System, website http://www.geod.nrcan.gc.ca/products-produits/gps_e.php (last accessed on September 16, 2008)

APPENDIX

Implementation Source Code

1. source.h
2. source.cpp (Version 1 – Read from text file)
3. source.cpp (Version 2 – Read from Rinex)
4. db4.h
5. db4.cpp
6. sink.h
7. sink.cpp (Version 1 – Writes to text file)
8. sink.cpp (Version 2 – Writes to Rinex file and computes postion)
9. main.cpp

Relevant GPS Toolkit Code

1. Bancroft.hpp
2. Bancroft.cpp

```
/*****************************************************************************
 Written by: Jorge Leon
 Date: September, 2008

 Name: source.h
 Reads a Rinex file, extracts the pseudorange values and makes them
 available to on its output ports
 *****************************************************************************/

struct source: sc_module {
 sc_in<bool> data_req;
 sc_out<bool> data_valid;
 sc_out<double> out_pr;
 sc_out<int> sat;
 sc_in_clk    clock;

 SC_CTOR(source)
  {
   SC_CTHREAD(entry, clock.pos());
  }

  void entry();
};
```

```
/*****************************************************************************
 Written by: Jorge Leon
 Date: September, 2008

 Name: source.cpp (version 1)
 Reads a text file, extracts the pseudorange values and makes them
 available on its output ports
 *****************************************************************************/
#include <iostream>
#include <iomanip>

#include "systemc.h"
#include "source.h"

using namespace std;

void source::entry()
{
  FILE *pseudorange;
  double tmp_val;

  pseudorange = fopen("pr1.txt", "r");
  data_valid.write(false);

  while(true)
  {
    wait_until(data_req.delayed() == true);
    if (fscanf(pseudorange,"%Lf \n",&tmp_val) == EOF)
    {
      wait(1000);
      cout << "End of Input Stream: Simulation Stops" << endl;
      sc_stop();
      break;
    }
    if (tmp_val < 20000000){ tmp_val = 0.0;}
    //cout << tmp_val << endl;
    out_pr.write(tmp_val);
    data_valid.write(true);
    wait_until(data_req.delayed() == false);
    data_valid.write(false);
    wait();
  }
}
```

```
/*****************************************************************************
 Written by: Jorge Leon
 Date: September, 2008

 Name: source.cpp (version 2)
 Reads a Rinex file, extracts the pseudorange values and makes them
 available on its output ports
 *****************************************************************************/
#include <iostream>
#include <iomanip>

#include "RinexObsBase.hpp"
#include "RinexObsHeader.hpp"
#include "RinexObsData.hpp"
#include "RinexObsStream.hpp"

#include "systemc.h"
#include "source.h"

using namespace std;
using namespace gpstk;

char* inobs = "ALBH1810.08O"; // file to read pseudoranges from

void source::entry()
{
 // Read pseudoranges from Rinex file
 data_valid.write(false);

 while(true)
 {
  wait_until(data_req.delayed() == true);

  // loop through satellites
  for(int satNum=1;satNum<=32;satNum++)
  {
   // Create the input file stream
   RinexObsStream rin(inobs);
   RinexObsData roe; //RINEX data object
   roe.obs.clear();

   // Loop over all data epochs
   while (rin >> roe)
   {
    wait_until(data_req.delayed() == true);
    SatID prn(satNum, SatID::systemGPS);
    RinexObsData::RinexSatMap::iterator pointer = roe.obs.find(prn);
    if ( pointer == roe.obs.end() ) {
     out_pr.write(0.0); // if no data for satellite prn, send zero
    }
    else {
     out_pr.write(roe.obs[prn][RinexObsHeader::P1].data);
    }
    sat.write(satNum);
    data_valid.write(true);
```

```
        wait_until(data_req.delayed() == false);
        data_valid.write(false);
        wait();
      }
    }
    wait(1000);
    cout << "End of Input Stream: Simulation Stops" << endl;
    sc_stop();
    break;
    }
  }
```

```
/*****************************************************************************
 Written by: Jorge Leon
 Date: September, 2008

 Name: db4.h
 Reads in pseudorange values, computes a discrete wavelet transform (DWT)
 based on Daubechies 4 wavelet, applies hard thresholding filtering, computes
 Inverse DWT and provides new pseudorange values on output port
*****************************************************************************/

struct db4: sc_module
{
  sc_in<double> in_pr;
  sc_in<int> in_sat;
  sc_out<int> out_sat;
  sc_out<double> out_pr;
  sc_in<bool> data_valid;
  sc_out<bool> data_req;
  sc_out<bool> data_out_ready;
  sc_in<bool> data_out_ack;
  sc_in_clk clock;

  SC_CTOR(db4)
  {
    SC_CTHREAD(dwtfilter, clock.pos());
  }

  void dwtfilter();
  void transform(double* a, const int n);
  void invTransform( double* a, const int n);
  void applyThresh( double* a, const int n, double t);
};
```

```
/*****************************************************************************
 Written by: Jorge Leon
 Date: September, 2008

 Name: db4.cpp
 Reads in pseudorange values, computes a discrete wavelet transform (DWT)
 based on Daubechies 4 wavelet, applies hard thresholding filtering, computes
 Inverse DWT and provides new pseudorange values on output port
 *****************************************************************************/
#include "systemc.h"
#include "math.h"
#include "db4.h"

double h0, h1, h2, h3;
double g0, g1, g2, g3;
double Ih0, Ih1, Ih2, Ih3;
double Ig0, Ig1, Ig2, Ig3;
double th = 500000;

/*********************************
 dwtfilter Module
*********************************/
void db4::dwtfilter()
{
  double sample[120];
  int satBuf[20];
  unsigned int index;
  unsigned int mark;
  unsigned int n;
  double* psample = NULL;
  bool sampleHasNoZeroValues;

  const double sqrt_3 = sqrt(3.0);
  const double denom = 4 * sqrt(2.0);

  h0 = (1 + sqrt_3)/denom;
  h1 = (3 + sqrt_3)/denom;
  h2 = (3 - sqrt_3)/denom;
  h3 = (1 - sqrt_3)/denom;
  g0  = h3;
  g1 = -h2;
  g2 =  h1;
  g3 = -h0;

  Ih0 = (3 - sqrt_3)/denom;
  Ih1 = (3 + sqrt_3)/denom;
  Ih2 = (1 + sqrt_3)/denom;
  Ih3 = (1 - sqrt_3)/denom;
  Ig0 = Ih3;
  Ig1 = -Ih2;
  Ig2 = Ih1;
  Ig3 = -Ih0;

  while(true)
  {
```

```
// Initialize control signals
data_req.write(false);
data_out_ready.write(false);
index = 0;
sampleHasNoZeroValues = true;

while( index < 120 )
{
  // Request data and read in samples
  data_req.write(true);
  wait_until(data_valid.delayed() == true);
  satBuf[index] = in_sat.read();
  sample[index] = in_pr.read();

  if (sample[index] == 0){
    sampleHasNoZeroValues = false;
  }

  index++;
  data_req.write(false);
  wait();
}

// Apply compute coefficient and apply de-noising
if (sampleHasNoZeroValues){
  n = 120;
  transform(sample, n);
  transform(sample, n/2);
  transform(sample, n/4);

  psample = &sample[n/8];
  applyThresh(psample, n/8, th);

  psample = &sample[n/4];
  applyThresh(psample, n/4, th);

  psample = &sample[n/2];
  applyThresh(psample, n/2, th);

  index = 0;
  // Perform Idwtfilter for 120 samples
  n = 120;
  invTransform(sample, n/4);
  invTransform(sample, n/2);
  invTransform(sample, n);
}

index = 0;
// Write Idwtfilter results on output port
while( index < 120)
{
  out_pr.write(sample[index]);
  out_sat.write(satBuf[index]);
  data_out_ready.write(true);
  wait_until(data_out_ack.delayed() == true);
```

```
        data_out_ready.write(false);
        index++;
        wait();
      }
      index = 0;
    } // end while
} // End of function dwtfilter

/**********************************
 Daubechies D4 transform
**********************************/
void db4::transform(double* a, const int n)
{
  if (n >= 4)
  {
      int i, j;
      const int half = n >> 1;
      double* tmp = new double[n];

      // Calculate wavelet coeffients
      for (i = 0, j = 0; j < n-3; j += 2, i++)
      {
         tmp[i]     = a[j]*h0 + a[j+1]*h1 + a[j+2]*h2 + a[j+3]*h3;
         tmp[i+half] = a[j]*g0 + a[j+1]*g1 + a[j+2]*g2 + a[j+3]*g3;
      }
      // Compensate for values at the end
      tmp[i]     = a[n-2]*h0 + a[n-1]*h1 + a[0]*h2 + a[1]*h3;
      tmp[i+half] = a[n-2]*g0 + a[n-1]*g1 + a[0]*g2 + a[1]*g3;

      // Save new values and delete temporary storage
      for (i = 0; i < n; i++)
      {
         a[i] = tmp[i];
      }
      delete [] tmp;
  } // end if
}

/**********************************
 Daubechies D4 inverse transform
**********************************/
void db4::invTransform( double* a, const int n )
{
  if (n >= 4)
  {
      int i, j;
      const int half = n >> 1;
      const int halfPlus1 = half + 1;
      double* tmp = new double[n];

      //     last smooth val  last coef.  first smooth  first coef
      tmp[0] = a[half-1]*Ih0 + a[n-1]*Ih1 + a[0]*Ih2 + a[half]*Ih3;
      tmp[1] = a[half-1]*Ig0 + a[n-1]*Ig1 + a[0]*Ig2 + a[half]*Ig3;

      for (i = 0, j = 2; i < half-1; i++)
```

```
    {
      //    smooth val    coef. val      smooth val    coef. val
      tmp[j++] = a[i]*lh0 + a[i+half]*lh1 + a[i+1]*lh2 + a[i+halfPlus1]*lh3;
      tmp[j++] = a[i]*lg0 + a[i+half]*lg1 + a[i+1]*lg2 + a[i+halfPlus1]*lg3;
    }
    for (i = 0; i < n; i++)
    {
      a[i] = tmp[i];
    }
    delete [] tmp;
  } // end if
}

/*********************************
 Apply fixed thresholding
*********************************/
void db4::applyThresh(double* a, const int n, double t)
{
  for (int i = 0; i < n; i++) {
    if ((a[i] > -t ) && (a[i] < t))
      a[i] = 0.0;
  }
}
```

```
/*************************************************************************

 Written by: Jorge Leon
 Date: September, 2008

 Name: Sink.h
 Reads in pseudorange values and updates original Rinex file with new
 pseudorange values. Computes the Bancroft position and generates Matlab
 m file with x and y values of ECEF position to make a scatter plot
 *************************************************************************

struct sink: sc_module
{
  sc_in<bool>  data_in_ready;
  sc_out<bool> data_in_ack;
  sc_in<double> in_pr;
  sc_in<int> in_sat;
  sc_in_clk    clock;

  FILE* cleanPR;

  SC_CTOR(sink)
  {
    SC_CTHREAD(entry, clock.pos());
  }

  void entry();
  void getPositions(int argc, char *argv[]);

  ~sink()
  {
  fclose( cleanPR );
  }
};
```

```
/*****************************************************************************
 Written by: Jorge Leon
 Date: September, 2008

 Name: Sink.cpp (version 1)
 Reads in values from input ports and writes them to a text file
 *****************************************************************************/
#include <iostream>
#include <iomanip>

#include "systemc.h"
#include "sink.h"

using namespace std;

void sink::entry()
{
  cleanPR = fopen("FilteredPseudoranges.txt","w");
  data_in_ack.write(false);

  while(true)
  {
     wait_until(data_in_ready.delayed() == true);
     fprintf(cleanPR,"%e  \n",in_pr.read());
     data_in_ack.write(true);
     wait_until(data_in_ready.delayed() == false);
     data_in_ack.write(false);
  }
}
```

```
/*****************************************************************************
 Written by: Jorge Leon
 Date: September, 2008

 Name: Sink.cpp (Version 2)
 Reads in pseudorange values and updates original Rinex file with new
 pseudorange values. Computes the Bancroft position and generates Matlab
 m file with x and y values of ECEF position to make a scatter plot
 *****************************************************************************/
#include <iostream>
#include <iomanip>

// Classes for handling observations RINEX files (data)
#include "RinexObsBase.hpp"
#include "RinexObsHeader.hpp"
#include "RinexObsData.hpp"
#include "RinexObsStream.hpp"

// First, let's include Standard Template Library classes
#include <string>
#include <vector>

// Classes for handling satellite navigation parameters RINEX files (ephemerides)
#include "RinexNavBase.hpp"
#include "RinexNavHeader.hpp"
#include "RinexNavData.hpp"
#include "RinexNavStream.hpp"

// Class for handling tropospheric models
#include "TropModel.hpp"

// Class for storing "broadcast-type" ephemerides
#include "GPSEphemerisStore.hpp"

#include "systemc.h"
#include "sink.h"

#include "Matrix.hpp"
#include "Bancroft.hpp"
#include "Vector.hpp"
#include "Position.hpp"

using namespace std;
using namespace gpstk;

char* inob = "ALBH1810.08O"; // base file to modify with new pseudoranges
char* inobnew = "ALBH1810new.08O"; // file with new pseudoranges
char* innav = "ALBH1810.08N"; // navigation file containing satellite position information

void sink::entry()
{
  RinexObsStream rin(inob); // Create the input file stream
  // Create the output file stream
  RinexObsStream rout(inobnew, ios::out|ios::trunc);
  RinexObsData roe; //RINEX data object
```

```
    RinexObsHeader head; //RINEX header obj

    rin >> head;
    rout.header = rin.header;
    rout << rout.header;

    double pr_array[32][120] = {0};
    int j=0;
    int k=0;
    int epochNum=0;
    int count=0;

    data_in_ack.write(false);

    while(true)
    {
       wait_until(data_in_ready.delayed() == true);
       for (j=1;j<=32;j++)
       {
          for (k=1;k<=120;k++)
          {
             wait_until(data_in_ready.delayed() == true);
             pr_array[j-1][k-1] = in_pr.read();
             data_in_ack.write(true);
             wait_until(data_in_ready.delayed() == false);
             data_in_ack.write(false);
          }
       }
       //Loop over all data epochs
       while (rin >> roe) {
          // loop through satellites
          for(int satNum=1;satNum<=32;satNum++){
             SatID prn(satNum, SatID::systemGPS);
             RinexObsData::RinexSatMap::iterator pointer = roe.obs.find(prn);
             if( pointer == roe.obs.end() ) ; // if no data for satellite prn, do nothing
             else
             {
                if (pr_array[satNum-1][epochNum]!=0)
                {
                   roe.obs[prn][RinexObsHeader::P1].data = pr_array[satNum-1][epochNum];
                }
             }
             count++;
             if(count>31){epochNum++;count=0;}
          }
          rout << roe;
       }
       char* fileNames[] = {" ", inobnew, innav};
       getPositions(3, fileNames);
    }
}

void sink::getPositions(int argc, char *argv[])
{
  // Declaration of objects for storing ephemerides
```

```
GPSEphemerisStore bcestore;

// Object for void-type tropospheric model (in case no meteorological RINEX
// is available)
ZeroTropModel noTropModel;

// Object for GG-type tropospheric model (Goad and Goodman, 1974)
GGTropModel ggTropModel;  // Default constructor => default values for model
// Pointer to one of the two available tropospheric models. It points to
// the void model by default
TropModel *tropModelPtr=&noTropModel;

try
{
   // Read nav file and store unique list of ephemeredes
   RinexNavStream rnffs(argv[2]);   // Open ephemerides data file
   RinexNavData rne;
   RinexNavHeader hdr;

   // Let's read the header
   rnffs >> hdr;

   // Storing the ephemeris in "bcstore"
   while (rnffs >> rne) bcestore.addEphemeris(rne);
   // Setting the criteria for looking up ephemeris
   bcestore.SearchNear();

   // Open and read the observation file one epoch at a time.
   // For each epoch, compute and print a position solution
   RinexObsStream roffs(argv[1]);   // Open observations data file
   // In order to throw exceptions, it is necessary to set the failbit
   roffs.exceptions(ios::failbit);

   RinexObsHeader roh;
   RinexObsData rod;

   // Let's read the header
   roffs >> roh;

   // Define Bancroft coordinates output file
   RinexObsStream xyz_ban("xyz_bancroft.m", ios::out|ios::trunc);
   bool isFirstPR = true;

   // Let's process all lines of observation data, one by one
   while (roffs >> rod)
   {
      // Apply editing criteria
      if (rod.epochFlag == 0 || rod.epochFlag == 1)  // Begin usable data
      {
         // Declare Xvt (position, velocity, time) object to store satellite positions
         Xvt myxvt;

         // Declare a matrix to store position values and an index for tracking rows
         Matrix<double> B(11,4);
         int index;
```

```
// Define initial position estimate for Bancroft position calculation
Vector<double> posEst(3);
posEst(0) = -2341332.634;
posEst(1) = -3539049.44676;
posEst(2) = 4745799.14508;

// Let's define the "it" iterator to visit the observations PRN map
// RinexSatMap is a map from SatID to RinexObsTypeMap:
//     std::map<SatID, RinexObsTypeMap>
RinexObsData::RinexSatMap::const_iterator it;

// This part gets the satellite positions and corresponding
// pseudoranges for the current epoch. They are stored into
// matrix B
for (it = rod.obs.begin(), index = 0; it!= rod.obs.end(); it++, index++)
{
   // RinexObsTypeMap is a map from RinexObsType to RinexDatum:
   //   std::map<RinexObsHeader::RinexObsType, RinexDatum>
   RinexObsData::RinexObsTypeMap otmap;

   // Let's define an iterator to visit the observations type map
   RinexObsData::RinexObsTypeMap::const_iterator itP1;

   // The "second" field of a RinexPrnMap (it) is a
   // RinexObsTypeMap (otmap)
   otmap = (*it).second;

   // Find a P1 observation inside the RinexObsTypeMap that
   // is "otmap"
   itP1 = otmap.find(RinexObsHeader::P1);

   // Store ECEF satellite postion data and Pseudoranges
   // into matrix B for Bancroft calculation
   if (itP1!=otmap.end())
   {
      SatID prn2((*it).first.id, SatID::systemGPS);
      myxvt = bcestore.getXvt(prn2,rod.time);
      B(index,0) = myxvt.x.operator [](0);
      B(index,1) = myxvt.x.operator [](1);
      B(index,2) = myxvt.x.operator [](2);
      B(index,3) = (*itP1).second.data;
   }
}
// compute BANCROFT position
Bancroft banSol;
int bresult;
bresult = banSol.Compute(B,posEst);

// Store position values in Matlab m file for plotting
xyz_ban << setprecision(12);
if (isFirstPR)
{
   xyz_ban << "x = " << posEst(0) << ";" << endl;
   xyz_ban << "y = " << posEst(1) << ";" << endl;
```

```
            isFirstPR = false;
          }
          else
          {
            xyz_ban << "x = cat(1,x," << posEst(0) << ");" << endl;
            xyz_ban << "y = cat(1,y," << posEst(1) << ");" << endl;
          }
        } // End usable data
      } // End loop through each epoch
      xyz_ban << "scatter(x,y)"; // write scatter plot command to m file (Bancroft)
    }
    catch(Exception& e)
    {
      cerr << e << endl;
    }
    catch (...)
    {
      cerr << "Caught an unexpected exception." << endl;
    }
    exit(0);
}
```

```
#pragma ident "$Id: $"

/**
 * @file Bancroft.hpp
 * Use Bancroft method to get an initial guess of GPS receiver's position
 */

#ifndef BANCROFT_HPP
#define BANCROFT_HPP

//============================================================================
//
//  This file is part of GPSTk, the GPS Toolkit.
//
//  The GPSTk is free software; you can redistribute it and/or modify
//  it under the terms of the GNU Lesser General Public License as published
//  by the Free Software Foundation; either version 2.1 of the License, or
//  any later version.
//
//  The GPSTk is distributed in the hope that it will be useful,
//  but WITHOUT ANY WARRANTY; without even the implied warranty of
//  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//  GNU Lesser General Public License for more details.
//
//  You should have received a copy of the GNU Lesser General Public
//  License along with GPSTk; if not, write to the Free Software Foundation,
//  Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
//
//  Dagoberto Salazar - gAGE ( http://www.gage.es ). 2006/2007
//
//============================================================================

#include <vector>
#include "Matrix.hpp"

using namespace std;
using namespace gpstk;

namespace gpstk
{
    /** @defgroup GPSsolutions GPS solution algorithms and Tropospheric
     *  models
     */

    //@{

    /** This class defines an algebraic algorithm to get an initial guess of
     *  GPS receiver's position given satellites' positions and pseudoranges.
     *
     * The algorithm is based on Bancroft's Method as presented in: Yang,
     * Ming & Kuo-Hwa Chen. "Performance Assessment of a Noniterative
     * Algorithm for Global Positioning System (GPS) Absolute
     * Positioning". Proc. Natl. Sci. Counc. ROC(A). Vol. 25, No. 2,
     * 2001. pp. 102-106.
     */
```

```
class Bancroft
{
public:

    /// Constructor
  Bancroft()
    throw(Exception) : SecondSolution(4,0.0)
  {
    testInput= true;
    ChooseOne = true;
    CloseTo = 6378137.0;
    minPRange = 15000000.0;
    maxPRange = 30000000.0;
    minRadius = 23000000.0;
    maxRadius = 29000000.0;
  };

    /** Compute an initial guess of GPS receiver's position , given
     *  satellites' positions and pseudoranges.
     *
     * @param Data    Matrix of data containing observation data in rows,
     *              one row per observation and complying with the
     *              following format:
     *
     *                  x y z P
     *
     *              Where x,y,z are satellite coordinates in an ECEF
     *              system and P is pseudorange (corrected as much as
     *              possible, specially from satellite clock errors),
     *              all expresed in meters.
     *
     * @param X     Vector of position solution, in meters. There may be
     *             another solution that may be accessed with vector
     *             "SecondSolution" if "ChooseOne" is set to "false".
     *
     * @return
     *   0  Ok,
     *  -1  Not enough good data
     *  -2  Singular problem
     */
  int Compute( Matrix<double>& Data,
          Vector<double>& X )
    throw(Exception);

    /** Another version of Compute method allowing calls with Matrix B
     *  being const.
     */
  int Compute( const Matrix<double>& Data,
          Vector<double>& X )
    throw(Exception);

    /** If true, the solution closest to CloseTo criterion will be chosen.
     *  If false, the two posible solutions will be provided.
     */
  bool ChooseOne;
```

```
      /** Criterion to decide which solution to choose. The algorithm will
       *  choose the solution closer to this value. By default, it is set
       *  to earth radius, in meters.
       */
      double CloseTo;

      /** If true (the default), the B input Matrix will be screened to get
       *  out suspicious data.
       *
       * It works with minPRange, maxPRange, minRadius and maxRadius to
       * pick up a set of "clean data". However, don't be too picky with
       * these parameters in order to leave room for different GNSS systems
       * and configurations. Anyway, Bancroft will give you just an
       * approximate position.
       */
      bool testInput;

      /// Minimum pseudorange value allowed for input data (in meters).
      double minPRange;

      /// Maximum pseudorange value allowed for input data (in meters).
      double maxPRange;


      /// Minimum allowed distance between Earth center and satellite
      /// position for input data (in meters).
      double minRadius;

      /// Maximum allowed distance between Earth center and satellite
      /// position for input data (in meters).
      double maxRadius;

      /** Vector<double> containing the estimated second position solution
       * (ECEF, meters), if ChooseOne is set to "false".
       */
      Vector<double> SecondSolution;

      /// Destructor.
      virtual ~Bancroft() throw() {};

   }; // end class Bancroft
      //@}
} // namespace gpstk
#endif  // BANCROFT_HPP
```

```
#pragma ident "$Id: Bancroft.cpp 1161 2008-03-27 17:16:22Z ckiesch $"

//============================================================================//  This file
is part of GPSTk, the GPS Toolkit.
//
//  The GPSTk is free software; you can redistribute it and/or modify
//  it under the terms of the GNU Lesser General Public License as published
//  by the Free Software Foundation; either version 2.1 of the License, or
//  any later version.
//
//  The GPSTk is distributed in the hope that it will be useful,
//  but WITHOUT ANY WARRANTY; without even the implied warranty of
//  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
//  GNU Lesser General Public License for more details.
//
//  You should have received a copy of the GNU Lesser General Public
//  License along with GPSTk; if not, write to the Free Software Foundation,
//  Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
//
//  Dagoberto Salazar - gAGE ( http://www.gage.es ). 2006/2007
//
//============================================================================

#include <cstdlib>           // for std::abs()
#include "Bancroft.hpp"
#include "MiscMath.hpp"
#include "Matrix.hpp"
#include "Vector.hpp"

using namespace std;
using namespace gpstk;

namespace gpstk
{

    /* 'Data'   Matrix of data containing observation data in rows, one
     *          row per observation and complying with this format:
     *                x y z P
     *          Where x,y,z are satellite coordinates in an ECEF system
     *          and P is pseudorange (corrected as much as possible,
     *          specially from satellite clock errors), all expresed
     *          in meters.
     *
     * 'X'      Vector of position solution, in meters. There may be
     *          another solution, that may be accessed with vector
     *          "SecondSolution" if "ChooseOne" is set to "false".
     *
     * Return values:
     *  0  Ok
     * -1  Not enough good data
     * -2  Singular problem
     */
  int Bancroft::Compute( Matrix<double>& Data,
                         Vector<double>& X )
    throw(Exception)
```

```
{
  try
  {

    int N = Data.rows();
    Matrix<double> B(0,4);    // Working matrix

      // Let's test the input data
    if( testInput )
    {

      double satRadius = 0.0;

        // Check each row of B Matrix
      for( int i=0; i < N; i++ )
      {
          // If Data(i,3) -> Pseudorange is NOT between the allowed
          // range, then drop line immediately
        if( !( (Data(i,3) >= minPRange) && (Data(i,3) <= maxPRange) ) )
        {
          continue;
        }

          // Let's compute distance between Earth center and
          // satellite position
        satRadius = RSS(Data(i,0), Data(i,1) , Data(i,2));

          // If satRadius is NOT between the allowed range, then drop
          // line immediately
        if( !( (satRadius >= minRadius) && (satRadius <= maxRadius) ) )
        {
          continue;
        }
          // If everything is ok so far, then extract the good
          // data row and add it to working matrix
        MatrixRowSlice<double> goodRow(Data,i);
        B = B && goodRow;
      }

      // Let's redefine "N" and check if we have enough data rows
      // left in a single step
    if( (N = B.rows()) < 4 )
    {
      return -1;  // We need at least 4 data rows
    }

  } // End of 'if( testInput )...'
  else
  {
      // No input filtering. Working matrix (B) and
      // input matrix (Data) are equal
    B = Data;
  }
```

```
Matrix<double> BT=transpose(B);
Matrix<double> BTBI(4,4), M(4,4,0.0);
Vector<double> aux(4), alpha(N), solution1(4), solution2(4);

  // Temporary storage for BT*B. It will be inverted later
BTBI = BT * B;

  // Let's try to invert BTB matrix
try
{
  BTBI = inverseChol( BTBI );
}
catch(...)
{
  return -2;
}

 // Now, let's compute alpha vector
for( int i=0; i < N; i++ )
{
    // First, fill auxiliar vector with corresponding satellite
    // position and pseudorange
  aux(0) = B(i,0);
  aux(1) = B(i,1);
  aux(2) = B(i,2);
  aux(3) = B(i,3);
  alpha(i) = 0.5 * Minkowski(aux, aux);
}

Vector<double> tau(N,1.0), BTBIBTtau(4), BTBIBTalpha(4);

BTBIBTtau = BTBI * BT * tau;
 BTBIBTalpha = BTBI * BT * alpha;

  // Now, let's find the coeficients of the second order-equation
double a(Minkowski(BTBIBTtau, BTBIBTtau));
double b(2.0 * (Minkowski(BTBIBTtau, BTBIBTalpha) - 1.0));
double c(Minkowski(BTBIBTalpha, BTBIBTalpha));
  // Calculate discriminant and exit if negative
double discriminant = b*b - 4.0 * a * c;
if (discriminant < 0.0)
{
  return -2;
}
  // Find possible DELTA values
double DELTA1 = ( -b + SQRT(discriminant) ) / ( 2.0 * a );
double DELTA2 = ( -b - SQRT(discriminant) ) / ( 2.0 * a );

  // We need to define M matrix
M(0,0) = 1.0;
M(1,1) = 1.0;
M(2,2) = 1.0;
M(3,3) = - 1.0;

  // Find possible position solutions with their implicit radii
```

```
      solution1 = M *  BTBI * ( BT * DELTA1 * tau + BT * alpha );
      double radius1(RSS(solution1(0), solution1(1), solution1(2)));
      solution2 = M *  BTBI * ( BT * DELTA2 * tau + BT * alpha );
      double radius2(RSS(solution2(0), solution2(1), solution2(2)));
        // Let's choose the right solution
      if ( ChooseOne )
      {
        if ( ABS(CloseTo-radius1) < ABS(CloseTo-radius2) )
        {
          X = solution1;
        }
        else
        {
          X = solution2;
        }
      }
      else
      {
          // Both solutions will be reported
        X = solution1;
        SecondSolution = solution2;
      }

      return 0;
    } // end of first "try"
    catch(Exception& e)
    {
      GPSTK_RETHROW(e);
    }
  } // end Bancroft::Compute()

    // Another version of Compute method to allow calling it with
    // const Matrix B.
  int Bancroft::Compute( const Matrix<double>& Data,
                         Vector<double>& X )
    throw(Exception)
  {
    Matrix<double> Datanoconst(Data);
    return Bancroft::Compute(Datanoconst, X);
  }
} // namespace gpstk
```

(2) BL-70-B7