TAMPEREEN TEKNILLINEN YLIOPISTO

Saku Rautiainen
Calculating GPS position using open source libraries
Bachelor of Science Thesis

Examiner: Helena Leppäkoski

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Bachelor's Degree Program in Telecommunications Electronics
RAUTIAINEN, SAKU:
Bachelor of Science Thesis
August 2011
Major: Digital Systems
Examiner: Helena Leppäkoski
Keywords: GPS, GPSTk, Position, Linux

GPS is the worlds first fully global navigation system. This thesis implements a GPS solution from Rinex navigation and observation files using Open source libraries. The GPSTk library was used for parsing Rinex files and getting satellite position. Corrections required for satellite position were implemented in this project, with data received from Rinex navigation files using object found in the GPSTk library. Parameters for calculating satellite position at GPS time was received from GPSTk library object which parsed Rinex Navigation file, while GPS time was received from similar object in GPSTk library which parsed Rinex Observation file. Satellite position was received from object in GPSTk library, after it had been given parameters and GPS time which had been corrected. Calculating receiver position from data in Rinex observation files using object found in GPSTk library and satellite position, was implemented with the help of AML++, AML++ provided the Matrix calculations and Matrix objects that algorithm required. The Graphical User Interface was implemented by using directly or inheriting objects found in the Qt library. The running environment was Ubuntu 10.04 and PC hardware. Program was implemented with the C++ programming language.

CONTENT

Abstract.....
Abbreviations, terms, and definitions 1
Symbols.....2
1. Introduction.....3
2. Position.....5
2.1 Positioning.....5
2.2 GPS 6
2.2.1 User Segment.....7
2.2.2 Control Segment.....7
2.2.3 Space Segment.....7
2.2.4 GPS conclusion.....7
3. Position Calculations.....8
3.1 Corrections utilized.....8
3.1.1 Transmission time.....9
3.1.2 Satellite clock difference.....9
3.1.3 ECEF rotation.....9
3.2 Least Squares method in solving receiver position.....10
3.3 From Cartesian to Geodetic coordinates.....13
4. Tools used.....14
4.1 Open source libraries.....14
4.1.1 Qt.....14
4.1.2 GPSTk.....15
4.1.3 AMLP++.....15
4.2 Rinex file format.....15
5. Program Design.....16
5.1 Class Diagram 16
5.2 Sequence Diagram 17
5.3 Design Features 18
5.4 Objects used in program.....18
5.4.1 Qt.....18
5.4.2 GPStk.....20
5.4.3 AML++.....20
5.4.4 Classes created for this project.....21
5.5 Summary of the program.....21
6. Verification 22
6.1 Testing positioning.....22
6.2 Testing Cartesian to Geodetic conversion.....23
7. Conclusion 24
8. Bibliography.....25

ABBREVIATIONS, TERMS, AND DEFINITIONS

Almanac	Data that holds the aproximate position of all satellites
Cartesian coordinates	Coordinates in a cartesian system, x-,y- and z-axises.
Clock Bias	Difference between clock value and another time reference
Ephemeris	Satellites data which pertains to satellites position in space
Geodetic coordinates	Coordinates in Geodetic system, latitude, longitude and altitude.
GPS	Global Position system, network of satellites and ground sations
GPL	General Public license, is a GNU project license
GPSTk	GPS library made by research group at Texas University, Austin
Latitude	Angular distance from the equator towards either North or South Pole
LGPL	Lesser General Public license, is a GNU project license
Linux	One of many Unix based operating systems
Longitude	Angular distance from the Greenwhich meridian line
PRN	Pseudo Random Number. Identifies which satellite is in question.
STL	Standard Template Library
Ubuntu	One of many Linux operating systems
WGS-84	World Geodetic System 1984, geodetic and geophysical modelling of earth

SYMBOLS

a_{f0}	Clock correction parameter	seconds
a_{f1}	Clock correction parameter	sec/sec
a_{f2}	Clock correction parameter	sec/sec^2
c	Speed of light	m/s
G	Geometric matrix	meters
h	Altitude measured from sea level	meters
N	Half of the distance between users coordinate and its mirror poin	meters
p_0	Pseudorange estimate	meters
p_c	Measured pseudorange, corrected with satellite clock difference	meters
$\boldsymbol{R}$	Satellite position vector	meters
t_{oc}	Ephemeris reference time	seconds
t	Gps time when signal was received by user	seconds
t_0	Gps time when signal left satellite	seconds
$\boldsymbol{V}$	Satellite velocity vector	m/s
$x^{(k)}$	Satellite coordinates	meters
x'	Satellite x coordinate after rotation	meters
x_k	X coordinate of satellite K	meters
x_0	Initial estimate for receiver coordinates (x,y,z)	meters
y'	Satellite y coordinate after rotation	meters
y_k	Y coordinate of satellite K	meters
z'	Satellite z coordinate after rotation	meters
z_k	Z coordinate of Satellite K	meters
Δt_{sv}	Satellite clock difference	seconds
Δt_r	Relastivistic time difference	seconds
δp	Pseudorange difference	meters
Ω_e	Earths angular velocity	rad/s
τ	Signal transmission time	seconds

1 INTRODUCTION

The original purpose of this project was to become familiar with GPSTk and use it for calculating position. Since then it evolved somewhat and the purpose became to calculate user position from rinex data by partly using GPSTk library and partly doing it on my own. Reason for the change was because GPSTk included ready-made examples which did the whole position calculating using its own objects.

Figure 1 holds the flow chart for the function of the program. Boxes with dashed lines describe functionalities which were created for this project where as those with solid lines are functionalities taken from external libraries. Names of the boxes follow the phases in the program. The first task in the program is to access Rinex data, both observation and navigation data and parse information out of them. From the navigation data we can create data structures that hold satellites' data. From the observation data we receive pseudorange measured at the time the signal was received by user. This time is given to satellite objects, which then calculate their own position using the time given to them. Together with the satellite position and pseudorange, we can calculate the User position. The last phase of the program consists of converting the Cartesian coordinates to Geodetic coordinates.

The Goal of this project is to learn how to use GPSTk and to perform GPS calculations outside of the MATLAB [1] environment.

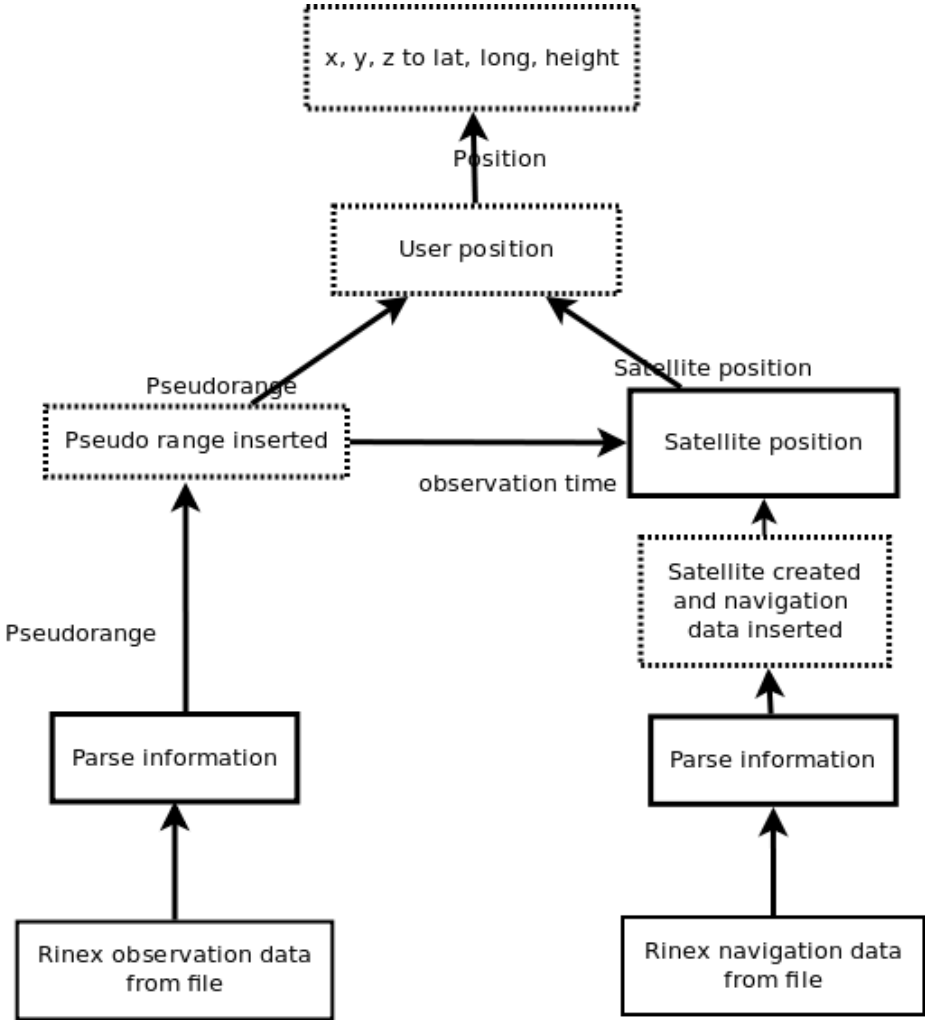

Figure 1: Diagram of the process. Solid line boxes represent ready-made parts of the project, while dash line boxes represent parts created for the project.

2 POSITION

Global Position System (GPS) is the first positioning system that was available to civilians with global coverage. Since its arrival, especially in the 21st century, many more position systems are emerging. Amongst the future's positioning systems is Galileo, which is a european project aimed to achieve more flexibility and accuracy. This does not lessen the usefulness of the GPS. Positioning is looked at from the perspective of GPS in this thesis.

Purpose of the GPS and positioning in general is to find the user's position in certain coordinate system.

2.1 Positioning

GPS receivers use signals sent by satellites to calculate their positions. There are four uknowns in these calculations; they are *x*,*y*,*z* coordinates and clock bias. Clock bias is the difference between the receiver's cheap inaccurate quartz clock chip and GPS system time. Four uknowns means that at least four satellite signals must be received. If there are more satellites the resulting position becomes more accurate due to more data being available.

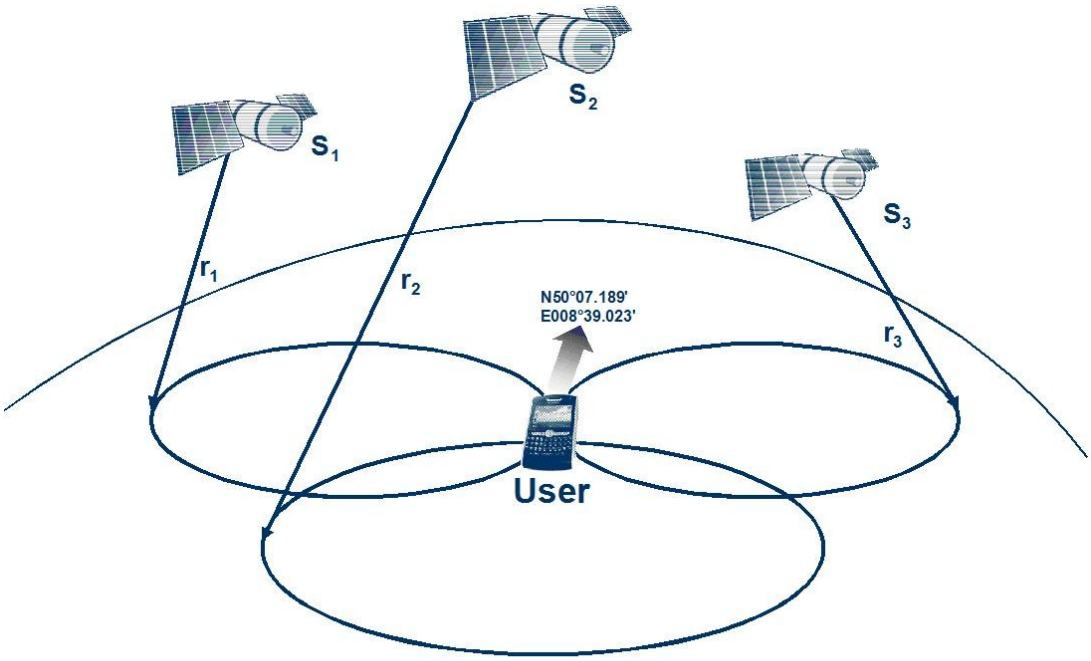

Figure 2: Simplified operation principle of GPS positioning system. [2]

From each signal, the receiver decodes the satellite's ephemeris data. Ephemeris holds all the necessary information about the satellite that sent the signal. This information includes orbital parameters, information about satellite health and various other data related to the satellite which sent the signal. From the ephemeris data, we can encode the time at which the signal was sent and the orbital parameters for calculating satellite position at that time. With these two and the time at which signal was received by the receiver, we can calculate user position [4].

From the time the signal takes to arrive to the receiver, we can calculate distance between each satellite and receiver. From the ephemeris data, more specifically from the orbital parameters, we calculate the satellite's position at the time the observation, pseudorange, was made.

Figure 2 shows a simplified operation principle of the GPS position. In this principle we have range instead of pseudoranges and thus no need to eliminate clock bias. Also, because of this only 3 satellites are needed for calculating user position.

2.2 GPS

Global Position System is composed of three different segments: the *User Segment, the Control* Segment and *the Space Segment*. Figure 3 shows all three and what their place is in the GPS system [4].

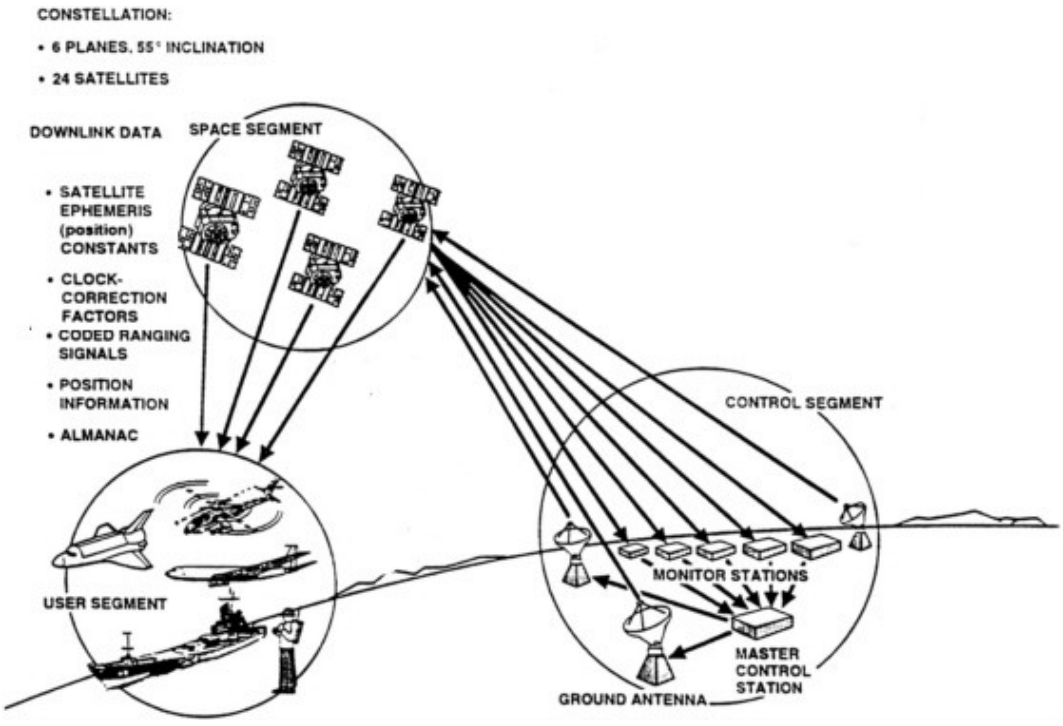

Figure 3: Gps system. Original picture modified [3].

2.2.1 User Segment

The User segment consists of all GPS receivers, hand-held or ones that are installed in vehicles. These GPS receivers hold in them an almanac, which tells the receiver where each satellite is aproximately located. The exact position for each satellite is calculated using each satellite's own ephemeris data. The GPS receivers receive the signal from satellites and using that signal and the information contained with in it, almanac and ephemeris, calculate their position using software and hardware [4].

2.2.2 Control Segment

The Control segment is the part of GPS system that monitors and makes corrections into the GPS. It is composed of 6 dedicated *Monitor Stations* and 4 dedicated ground antennas which monitor satellite positions and their signals. These in return pass their observations to the *Master Control Station*, or to its *Alternate Master Control Station*. Master Control Station sends back to the satellites clock adjustments and ephemeris constants calculated from the data observed by Monitor Stations[4]. There are also several shared *Monitor Stations* in existence [5].

2.2.3 Space Segment

Space segment consists of 24 satellites that are about the size of a car. All of them have been launched from Cape Canaveral, Florida. Circulation time for a satellite in GPS system is 12 hours, that is the time it takes for it to go around the earth once. They are divided into 6 orbital planes of 4 satellites each. Minimum number of these satellites needed for 3D location is 4; that where we know latitude, longitude and height or x, y , and z [4].

2.2.4 GPS conclusion

Table 1: GPS system facts.[6]

Full coverage	24 satellites in orbit
3D positioning	4 satellites
Accuracy without assist	3 meters
Current orbital coverage	30 satellites in orbit
Control stations	1 Master Control Station and 5 control stations
Accuracy with assist (DGPS)	To few centimeters

3 POSITION CALCULATIONS

There are many ways to calculate user position. In this chapter, the method and corrections utilized in the program written for this thesis are explained.

3.1 Corrections

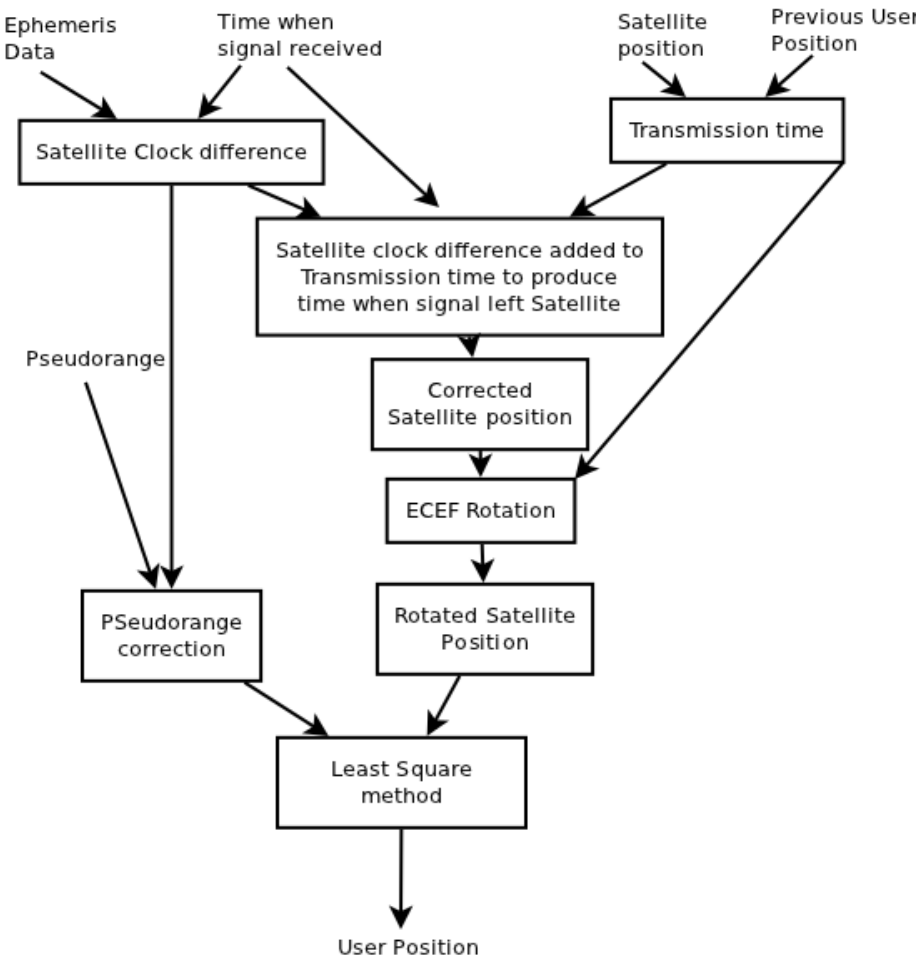

Figure 4: Corrections

There are some corrections that had to be utilized in this program due to not receiving the time at which signal left from satellite or satellite clock difference. This problem is due to the Rinex file specification. So, we have to take satellite clock difference, relastivistic effect in space and the clock bias of the satellite, into account. Second correction to the time is applying transmission time, with this we will get the time when signal left satellite. This time is needed for getting satellite position. Third

correction is to rotate satellite coordinates in ECEF coordinate system. Figure 4 summarizes the use of these corrections.

3.1.1 Transmission time

Transmission time is the time between when the signal left from the satellite and when it was received by the user. If a coarse estimate of user position is know, we will utilize the following formula

$$\tau^{(k)} = \frac{\|x^{(k)} - x_0\|}{c} \quad . \tag{1}$$

Variables c, x_0, and $x^{(k)}$ are speed of light, coarse estimate of user position and satellite position respectively. Value used for speed of light utilized in these calculations is 299792458. If no user position is know, we aproximate that $\tau^{(k)} = 0.07$ s. Transmission time is required for calculating the time when signal left the satellite. Over time as we calculate $\tau^{(k)}$ again and again, we will get more accurate position [7].

3.1.2 Satellite Clock difference

Clock difference has to be taken into account when calculating satellite position, that is the relastivistic effects and satellite clock bias. It is also utilized in calculating correction to pseudorange. Receivers clock difference is dealt with the Least Square Solution [9].

$$\begin{aligned} \Delta t_{sv} &= a_{f0} + a_{f1} * (t - t_{oc}) + a_{f2} * (t - t_{oc})^2 + \Delta t_r \\ &\text{where } \Delta t_r = \frac{2\vec{R} \circ \vec{V}}{c^2} \end{aligned} \tag{2}$$

Δt_r is the relastivistic effect in the satellites clock. t is the time when signal was received, a_{f0}, a_{f1}, and a_{f2} are variables that are delivered from the satellite in the ephemeris data, as is t_{oc}. **R** and **V** are the position and velocity vectors of satellite, respectively [8].

3.1.3 ECEF rotation

Due to ECEF coordinate system rotating as the earth rotates, we need to rotate the satellites coordinates to the point in time when the signal is received by the user[8].

$$\begin{aligned} \theta &= \dot{\Omega}_e * (t - t_0) \\ x' &= x * \cos(\theta) - y * \sin(\theta) \\ y' &= x * \sin(\theta) + y * \cos(\theta) \\ z' &= z \end{aligned} \tag{3}$$

In this equation, Ω_e is the angular velocity of earth, value for that is 0.00007292115. x, y, and z are the satellite coordinates before rotation and x', y', and z' after rotation. t-t_0 is the time between signal leaving satellite and being received by

the user, with t when it was received by the user and t_0 when it was sent by the satellite. In the program this was replaced by using τ, since it also describes the transmission time.

3.2 Least Square method in solving position

Since calculating the satellite position is provided by the GPSTk library, I will not go into explaining it. Rather I will explain how from the satellite's position and pseudorange to receiver we can calculate the receiver position. This method is called the least squares method. The Algorithm for the least squares method can be found in the lecture notes [9].

Step 0: Obtain data, in this case pseudorange measurements and satellite position. Parameters $p_c^{(1)}...p_c^{(k)}$ are pseudoranges, while parameters $x^{(1)}...x^{(k)}$, $y^{(1)}...y^{(k)}$, and $z^{(1)}...z^{(k)}$ are estimates for satellite positions we got with equation (3). K is the number of the satellites.

$$\begin{bmatrix} p_c^{(1)} \\ \vdots \\ p_c^{(k)} \end{bmatrix} \text{ Pseudorange } \quad \begin{bmatrix} x^{(1)} & y^{(1)} & z^{(1)} \\ \vdots & \vdots & \vdots \\ x^{(k)} & y^{(k)} & z^{(k)} \end{bmatrix} \text{ Satellite positions} \tag{4}$$

Step 1: Choose initial estimate of the receiver position, $\mathbf{x_0}$, and receiver clock bias b_0. Usually on earth if no better info is avalaible, we use the following:

$$\begin{bmatrix} \boldsymbol{x_0} \\ b_0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T \ . \tag{5}$$

Step 2: Calculate distance estimates for all available satellites k= 1,..,K. That is, calculating distance between satellite position and our estimate for users position.

$$r_0^{(k)} = \left|\left| \boldsymbol{x^{(k)}} - \boldsymbol{x_0} \right|\right| = \sqrt{(x^{(k)} - x_0)^2 - (y^{(k)} - y_0)^2 - (z^{(k)} - z_0)^2} \tag{6}$$

Step 3: Calculate pseudorange estimates (k= 1,..,K), p_0.

$$p_0^{(k)} = r_0^{(k)} + b_0 \tag{7}$$

Step 4: Calculate pseudorange differences (k= 1,..,K.), δp_0. That is, difference between corrected pseudorange and our estimated pseudorange. Corrected pseudorange is the one that has the time correction already done onto it. Where as pseudorange estimate has the bias correction in it and as we iterate it gets more accurate.

$$\delta\, p_0^{(k)} = p_c^{(k)} - p_0^{(k)} \tag{8}$$

Step 5: Make a Geometry matrix, $\boldsymbol{G}$. Note that in a Geometry matrix, the absolute value of an element is smaller or equal to one.

$$\boldsymbol{G}=\begin{bmatrix} \frac{x_0-x^{(1)}}{r_0^{(1)}} & \frac{y_0-y^{(1)}}{r_0^{(1)}} & \frac{z_0-z^{(1)}}{r_0^{(1)}} & 1 \\ \vdots & \vdots & \vdots & 1 \\ \frac{x_0-x^{(K)}}{r_0^{(K)}} & \frac{y_0-y^{(K)}}{r_0^{(K)}} & \frac{z_0-z^{(K)}}{r_0^{(K)}} & 1 \end{bmatrix} \tag{9}$$

Step 6: Calculate corrections to estimates using the Least squares solution. $\delta\boldsymbol{x}$ is a vector holding corrections to position and δb is correction to clock bias.

$$\begin{bmatrix} \boldsymbol{\delta}\hat{\boldsymbol{x}} \\ \delta\hat{b} \end{bmatrix}=(\boldsymbol{G}^T\boldsymbol{G})^{-1}\boldsymbol{G}^T\boldsymbol{\delta}\boldsymbol{p} \tag{10}$$

Step 7: Apply corrections to estimates.

$$\begin{aligned} \hat{\boldsymbol{x}} &= \boldsymbol{x_0}+\delta\hat{\boldsymbol{x}} \\ \hat{b} &= b_0+\delta\hat{b} \end{aligned} \tag{11}$$

Step 8: Check if stopping criteria is fulfilled. We can set how small the correction must be. If its too large, then we iterate all over again, from Step 2 on. Equation (12) is used to check stopping criteria by taking the norm of the position change.

$$||\delta\hat{\boldsymbol{x}}||=\sqrt{\delta x^2+\delta y^2+\delta z^2} \tag{12}$$

Lecture notes [8] include numerical example for showing how least squares solution works. Least Square solution is summed in figure 5.

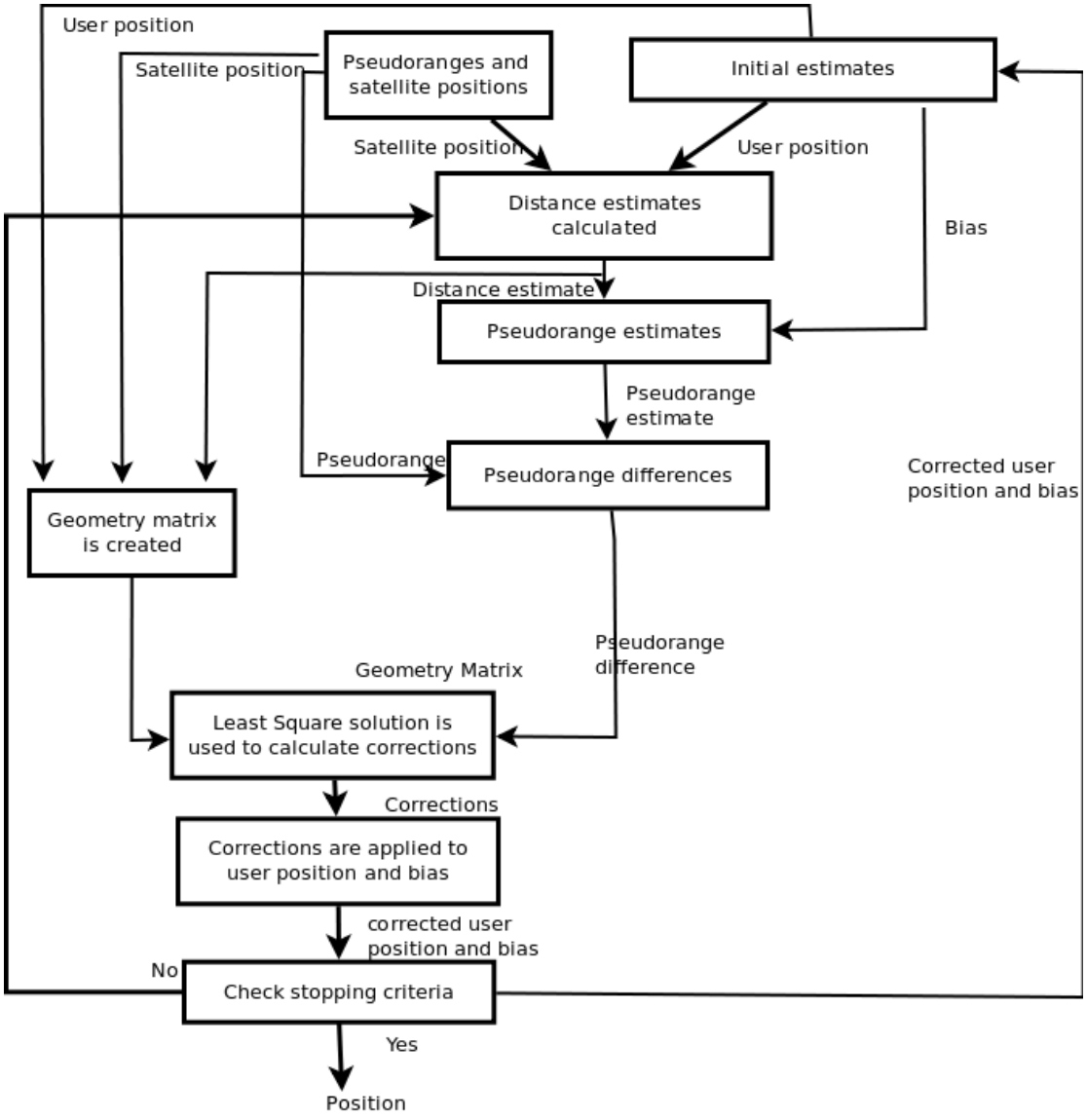

Figure 5: Flowchart displaying the process of using Least Square method.

3.3 From Cartesian to Geodetic coordinates

The Least Squares solution gives us Cartesian cordinates (x,y,z), and inorder for the position to be usefull data to a normal user, it needs to be in a Geodetic coordinate system. The Algorithm for this conversion can be found in lecture notes [10].

$$\begin{aligned} longitude &= atan2(x, y) \\ degrees &= longitude * 180 * pi \\ p &= \sqrt{x^2 + y^2} \end{aligned} \tag{13}$$

Atan2 is a variation of the arctangent function. Format used for it here is the one used in matlab. AML++ library which was utilized in the program also includes a atan2 function.

Calculating latitude and height requires iteration. Here we can also set a parameter for how much change we allow in latitude before we stop iteration. Or do a certain number of iterations only. First we calculate N and h which are required for calculating latitude. Variables a and e^2 are parameters in WGS-84(World Geodetic System 1984) system, a is 6378137 and e^2 is 0.0067[10]. Value for latitude is at the beginning 0. h is the altitude.

$$N = \frac{a}{\sqrt{1 - e^2 * \sin(latitude)^2}} \tag{14}$$

$$h = \frac{p}{\cos(latitude)} - N \tag{15}$$

Then we calculate latitude.

$$latitude = atan\left(\frac{z}{p} * \left(\frac{1}{\left(\frac{1 - e^2 * N}{N + h}\right)}\right)\right) \tag{16}$$

After having done so, we calculate N and h again and then latitude, until we are satisfied that the change in latitude is small enough.

A Numerical example can found in lecture notes [11] for this conversion.

4 TOOLS USED

The Tools utilized in this program were various open source libraries. The Software itself was created in a Linux OS, Ubuntu version 10.04, and it was meant to be used in Linux.

4.1 Open source libraries

Various open source libraries were utilized in creating this project. Most of these libraries fell under either GNU GPL (General Public License) or GNU LGPL (Lesser General Public License) licencees. They are open source licences for programs and they define what a program can be used for [12].

4.1.1 Qt

Qt is an open source library for the purpose of making graphical user interfaces [13]. It was first created by Trolltech, a software company based in Norway. It was bought by Nokia in 2006 and has since then been made into open source. It is available for all popular desktop OS's and some mobile OS's. Qt is available under GNU GPL 3.0.

The Qt library was utilized to a small extent in this project. It was used mainly for creating the graphical user interface. In Figure 6 is a screenshot of the graphical user interface created for the program. Instead of using a text based interface, a graphical user interface is easier for displaying and browsing data [13].

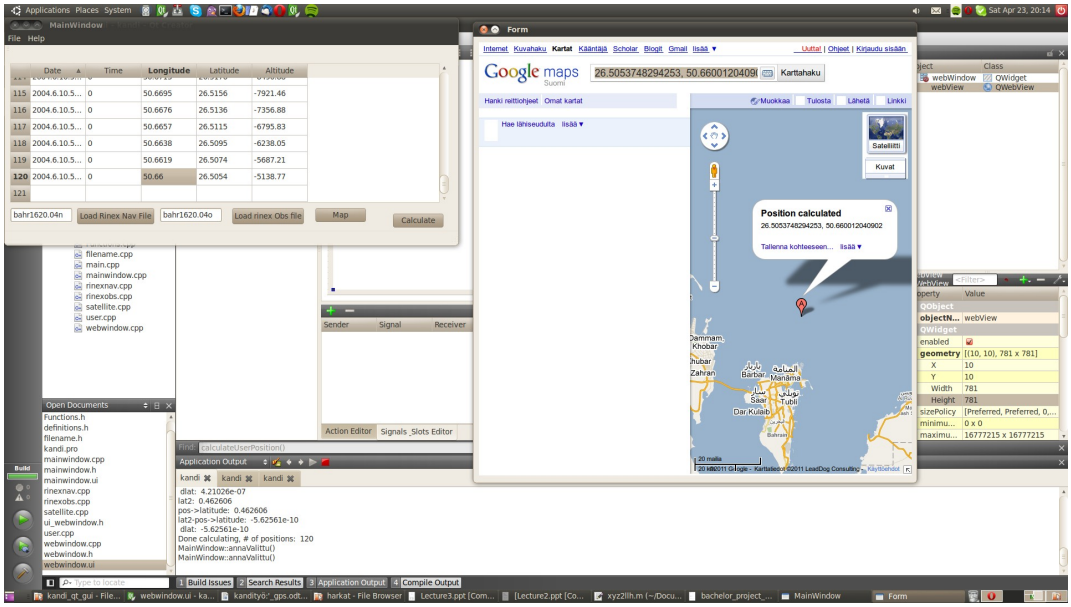

Figure 6: Software running on on the environment, Ubuntu OS.

4.1.2 GPSTk

The GPSTk library is an open source library that provides a variety of tools for handling GPS data. It is meant for researchers so that they can concentrate on research instead of writing code. It was created at the University of Austin, Texas. GPSTk is available under GNU LGPL. GPSTk is available for Linux and Windows.

GPSTk provides all the tools for accessing Rinex data and computing position from that data. These tools also include objects which take atmospheric conditions into account.

The Latest version of the GPSTk is from 2008, but support continues to be strong [14].

4.1.3 AML++

AML++ is a matrix library that mimics most Matlab and Octave mathematical functions and commands [15]. Like GPSTk, AML++ was meant for researchers for the same reasons, to do research instead of programming. AML++ is released under GNU GPL. AML++ is fairly new, latest version 0.3.2, which is the one used in the program, was released in 11.01.2010 [15].

4.2 Rinex file format

Receiver Independent Exchange format Rinex is utilized in positioning to exchange navigation, observation and meterogical data no matter what the receiver is. Rinex is in human readable, text format. This project utilized Rinex navigation and observation files in giving data to the program for calculations. It was originally developed by the Astronomical Institute of the University of Berne for massive GPS data gathering campaign, which involved 60 different receivers. Observation file has all the 'observations' aka pseudorange, time frequenzy etc. made by the receiver. Navigation file has all the data, orbital parameters necessary for calculating satellite position. [16]

5 PROGRAM DESIGN

This chapter will illustrate the design of the program and how it functions.

5.1 Class Diagram

Figure 7 holds the class diagram of the program. The Class Diagram includes all the classes created for the project. These classes can be divided into three categories, user interface, data handling and position.

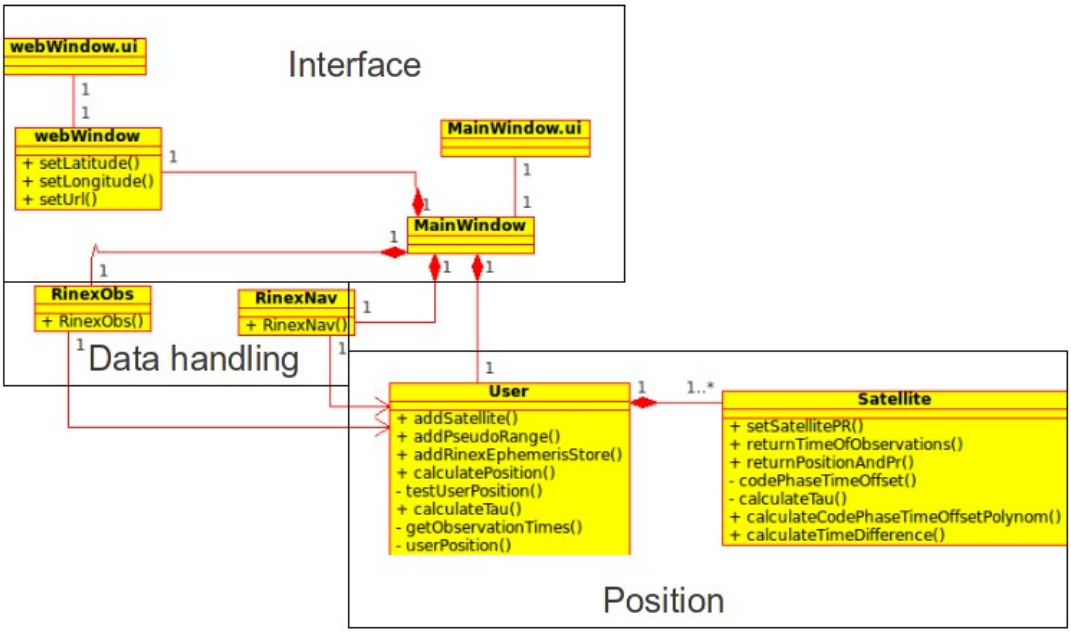

Figure 7: Class diagram of program.

The Interface category includes the *MainWindow* and *webWindow* classes, webWindow.ui and and mainWindow.ui. Files with ui extension are Qt's own graphical files for creating a user interface by dropping objects to the interface. *MainWindow* class is the functionality behind the interface; it includes all the functionality of the program. It owns all the other classes with the exception of the Satellite class. The w*ebWindow* class's only functionality is showing Google Maps webpage with the coordinate point in it.

The Data handling category includes two classes: R*inexNav* and *RinexObs*. These are used for handling all the data. The *RinexNav* object's purpose is to get the navigation data from the rinex navigation file. From this navigation data *User* creates *rinexEphemerisStore* object, which belongs to GPSTk library, and stores in that all of the satellite's navigation data. The Pointer to this *rinexEphemerisStore* is passed onto *Satellite* when *User* creates it.

The *RinexObs* object's purpose is to get the observation data from the rinex observation file and pass this unto the *User* object. From this observation data , User object requests from each satellite their position at the GPS time when a signal left satellite. Corrections are made into the time when signal was received by the user before requesting position using equations (1)-(3), with these corrections can the time when signal left the satellite be calculated. This observation and satellite position is saved into the S*atellite* using map, with time being key. Map in this case is a C++ STL (Standard Template Library) container. Map can be accessed using a key and data can be stored for that specific key. Maps can be accessed either through using the key or using iterators [17].

The Last category is position, this includes all the objects necessary to calculate position: *User* and *Satellite*. *User* object owns all *Satellite* objects and the original *rinexEphemerisStore* object which stores the satellite's navigation data. User stores all the satellites and other data needed for calculations in map containers, where they can be retrieved easily and quickly.

The *Satellite* object stores satellite's position and distance to *User* at given times. It also includes PRN (Pseudo Random Number) with which to identify the satellite in question. This is done mostly using maps as containers.

5.2 Sequence Diagram

Figure 8 holds the sequence diagram of the program.

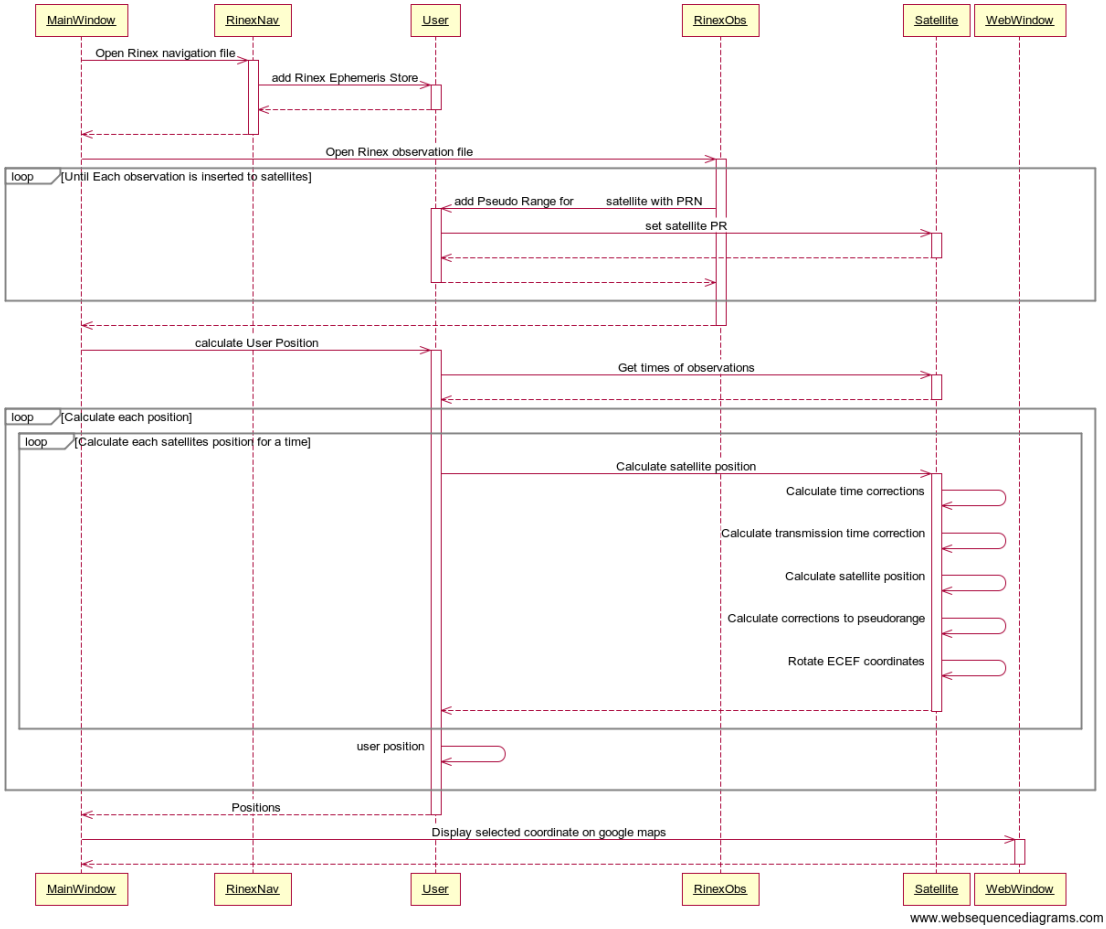

Figure 8: Sequence diagram of program.

The Program starts with the opening of the Rinex navigation file using the constructor for the *RinexNav* object, this in turn creates from the Rinex Navigation file *RinexEphemerisStore* objects and gives pointer from that object to User.

Next the program opens the Rinex observations file using the *RinexObs* objects, this passes each observation, pseudorange, to the User object along with the time when signal was received by receiver and identifying PRN, satellites id. User creates from this observation a satellite if it does not already exist, if it does then it merely gives the satellite the pseudorange and GPS time. Satellite is only capable of holding one pseudorange per GPS time. This is done until each observation has been processed.

After this, *MainWindow* requests positions from *User*, this in turn requests the GPS times at which observations were made for each satelite. For each GPS time, it requests from *Satellites* the position and pseudorange at that GPS time. Before requesting the satellite position for the time the observation was made, corrections must be calculated for the correct time when the signal left the satellite using equations (1)-(2). After this the coordinates must be rotated using equation(3)

Each time there are four or more satellites, using those satellite position and pseudorange to *User*, user position is calculated using equations (4)-(12). For each position it computes, it is converted to geodetic coordinate system using equations (13)-(16). Once all possible position solutions have been calculated for observation set and satellite coordinate sets, User will pass them back to the *MainWindow*.

From *MainWindow* each cordinate set can be shown in a Google maps using *WebWindow*.

5.3 Design Features

Due to limitations of the AML++ library, matrix sizes can not be determined during run. They have to be set before compilation. Because of this, Matrices had to be created along with their calculations for each case.

5.4 Classes used in program

5.4.1 Qt

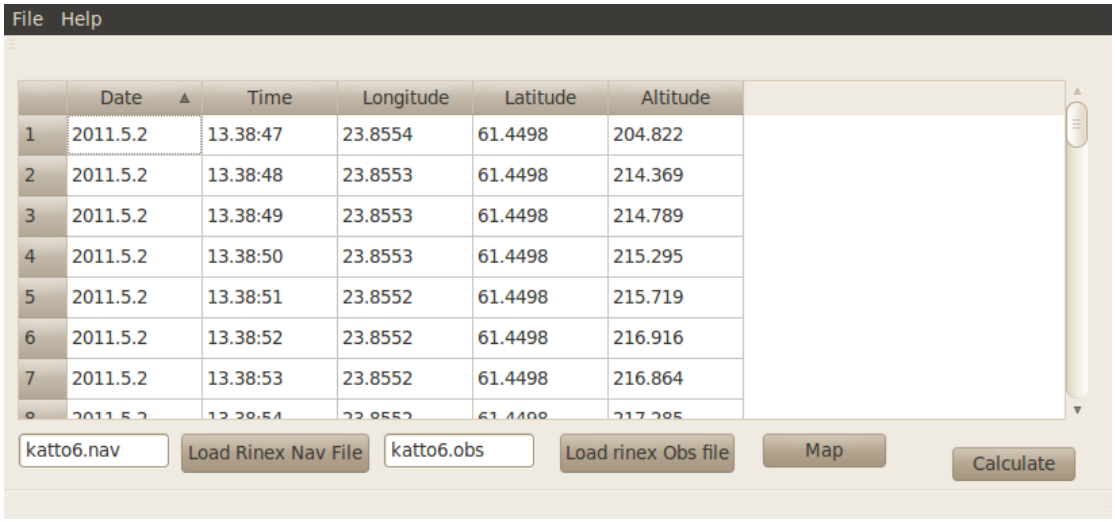

Figure 9: GUI screenshot.

Figure 9 holds the main gui while figure 10 holds the *webWindow* gui, which displays coordinate in Google maps.

The Main gui was created by using Qt's own *QMainWindow* class. This class provides the basic gui where the developer either graphically or in the code places what he needs. In this case it includes one button for calculating position from the data. The Data was displayed in *QStandardItemModel*, which is part of Qt's MVC (Model/View/Control). *QStandardItemModel* itself was inside *QTableView*. *QStandardItemModel* stores data in *QstandardItem*'s.

About dialog and *webWindow* were created with *QDialog* class. About dialog is window for showing creators name, student id and what program is utilized for. Unlike *webWindow*, *About* dialog was not inherited and thus does not posses any functionality of its own. *QWebView* was used in webwindow object to display selected coordinates in Google Maps [1]. Figure 9 shows *QDialog*, wich holds in it *QwebView*. Figure 10 shows the ui created with webWindow class. Table 2 summarizes the Qt objects used in the program.

Table 2: Qt classes utilized in program[1]

Class	Purpose
QMainWindow	Main gui.
QStandardItemModel	Class which holds and displays data related to position.
QTableView	Qt table for holding items
QstandardItem	Qt item for holding individual data in it and displaying data in a table.
QDialog	*About* and *webWindow* windows were created with this class
QWebView	Object which displays webpages in it.

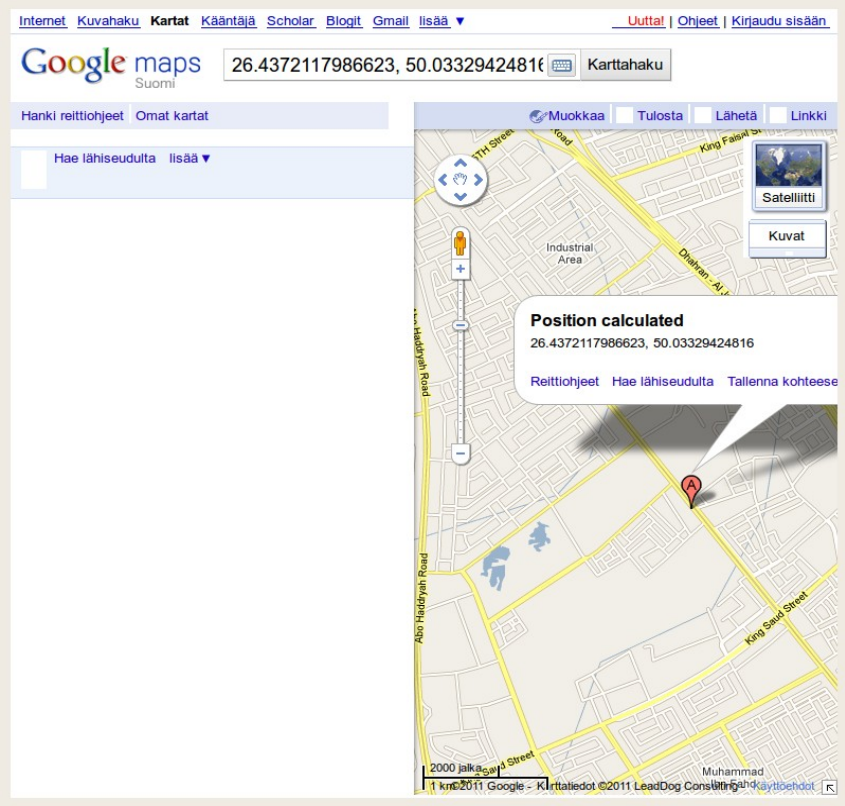

Figure 10: Coordinate point shown in google maps loaded onto QwebView object.

5.4.2 GPStk

For reading the Rinex Navigation files, *rinexEphemerisStore* object from GPStk was used. By inserting the file name into rinexEphemerisStore, it reads the file, parses information from it and with its member functions the developer can request different kinds of data, making Rinex Navigation file very easily read. The Data in this programs case is the satellite position at the GPS times when the signal was received.

Reading the Rinex Observation file was done by using a combination of objects. File header was read into *RinexObsHeader*, while data was read into *RinexObsData* entry by entry. Each entry was then searched for pseudorange, GPS time and satellite id, PRN. These three were given to User object which in turn would insert pseudo range and time for the *Satellite* object.

5.4.3 AML++

From AML++ library, Real, RealVector, and RealMatrix objects were used. RealVector is a Vector for numbers, similar to MATLAB's [16] single line Matrix. Real is the equilevant of C++ double type. RealVector andRealMatrix are like matlab vectors and matrices; their member functions are very similar [3].

Also, we had to implement $(G^T * G)^{-1} * G^T * drho$ in our calculations. In AML++ library a developer can use the *lsolve* function for this purpose [3].

5.4.3 Classes created for this project

Various classes were written for this program, that are utilized as objects. A Total of four classes apart from the *MainWindow* and *webWindow* class were created. The four classes are *RinexNav*, *RinexObs*, *User*, and *Satellite*.

MainWindow is the class behind the ui, adding more functionality to it. Same thing with the *webWindow*, although much less so. *WebWindow*'s purpose is to display coordinate points in a manner that is more useful to a normal user, on a map.

RinexNav's purpose is to open the Rinex navigation file using GPSTk objects, and pass from there the navigation data to the User object. *RinexNav*'s purpose is to open the rinex observations file using GPSTk objects, and pass from there the observation data to a *Satellite* object through the User object.

The *User* object is meant to hold a specific user's satellites from which this user received signals and from those signals aka pseudoranges and satellite position coordinates calculate the user's position at times. *Satellite* is meant to hold the GPS satellites orbital parameters, navigational data, and observations made by receiver from having received signal. Using these the *Satellite* calculates its position in orbit at the time observations were made, and passes these back to *User*.

5.5. Summary of the program

Table 4: Summary of program, code lines include also those for ui desgin forms, even though they are xml created by Qt Designer

Number of code lines	2179
Number of xml code lines	203
Number of graphical windows	3
Number of classes created	6

6 VERIFICATION

The Program was verified with a test data set. These included rinex measurements taken by the Department of Computer Systems.

6.1 Testing positioning

Positioning was tested with Rinex files that were received from the TUT Department of Computer Systems and comparison results were from the example program written in MATLAB at the TUT Department of Computer Systems. This testing shows very small inconsistencies in results. Table 5 holds results and its comparison solution.

Table 5: Data in both cases is at GPS second 136044. Example solution is from example program written in MATLAB in TUT Department of Computer Systems. Data was measured from GPS antenna at the roof of Tietotalo.

coordinates	Data source	*x(m)*	*y(m)*	*z(m)*
Example[8]	Example data	2795201,25	1236061,48	5579648,39
Project	Example data	2795201.89	1236059.95	5579647.60

Testing of the Least Square solution proved to be very accurate, at least in comparison to reference solution[9]. Table 6 holds results and its comparison solution.

Table 6: Data and example solution is from lecture notes[9]. Project solutions were received using testUserPosition function(). Results are after one iteration.

coordinates	Data source	*x(m)*	*y(m)*	*z(m)*
Example[8]	Example data	$3.0447*10^6$	$0\ 9598\ *10^6$	$6.8790\ *10^6$
Project	Example data	$3.04467\ *10^6$	$0.959844\ *10^6$	$6.87903\ *10^6$

6.2 Testing Cartesian to Geodetic conversion

Testing the Cartesian to Geodectic conversion also proved to be accurate, in comparison to example[9]. That is converting from *x,y,z* coordinates to *altitude*, *latitude*, *longitude* coordinates. Table 7 holds results and its comparison solution.

Table 7: Testing the Cartesian to Geodetic conversion with the xyz2llhTest.cc file, comparison data is from example in lecture notes[9]. Xyz2llhTest.cc implements the same conversion as found in project, except with console interface for using it. X coordinate is 2798340.2052 m, y coordinate is 1216752.9506 m, and z coordinate is 5582405.2377 m.

coordinates	*longitude(°)*	*latitude(°)*	*altitude(m)*
Example[9]	23,5	61,5	299,99
Xyz2Test.cc	23.50000003	61.50000005	300.9810477

7 CONCLUSION

As part of this project, I designed and created a program that is supposed to open Rinex, navigation and observation, files and calculate a GPS position from them. This was done through the use of various open source libraries. Goal of this project was to learn how to use these libraries, especially GPSTk and to create a working program. This goal was achieved as demonstrated by the verification chapter.

Next steps in improving this program lie in implementing 2d positioning, which should be fairly easy. Also, implementing data processing which eliminates any internal consistencies in data might also be useful. UI wise, it might be helpfull to implement such a mapping which allows for multiple points in the map.

Table 8: Facts about the project

Platform	Linux
Programming language used	C++
Open Source libraries used	Qt, GPSTk, AML++
Compiler	gcc version 4.4.3 (Ubuntu 4.4.3-4ubuntu5)
Number of files	19(exluding test files)
Number of code lines	2185

8 BIBLIOGRAPHY

[1] Mathworks inc., MATLAB, http://www.mathworks.com/, 19.05.2011

[2] FIDIS, Figure about satellite positioning, http://www.fidis.net/resources/deliverables/mobility-and-identity/int-d1110001/doc/10/, referenced 29.04.2011

[3] The National Academies Press, Figure about GPS, http://www.nap.edu/openbook.php?record_id=9254&page=6, referenced 29.04.2011

[4] Alfred Leick, GPS Satellite Surveying, 352 pages, first edition, 1990

[5] Aerospace Corporation, Elements of GPS, http://www.aero.org/education/primers/gps/elements.html, referenced 14.05.2011

[6] US Goverment, GPS website, http://www.gps.gov/, referenced 29.04.2011

[7] Helena Leppäkoski , Position Estimation with Pseudoranges, 4 pages, referenced 01.05.2011

[8] US Goverment, GPS Interface Specification, ,http://www.gps.gov/technical/icwg/IS-GPS-200E.pdf 185 pages, referenced 01.05.2011

[9] TUT, TKT-2536 Satelliittipaikannuksen perusteet, course lecture 2 material, 2010, 15 pages, referenced 01.05.2011

[10] NIMA, National Imagery and Mapping Agency Technical Report, referenced 20.08.2011

[11] TUT, TKT-2536 Satelliittipaikannuksen perusteet, course lecture 4 material, 2010, 17 pages, referenced 01.05.2011

[12] GNU, GNU licences document, http://www.gnu.org/licenses/licenses.html, referenced 29.04.2011

[13] Nokia inc., QT Document, http://doc.qt.nokia.com/4.7/index.html, referenced 29.04.2011

[14] The University of Texas at Austin, GPSTk document, http://www.gpstk.org/bin/view/Documentation/WebHome, referenced 29.04.2011

[15] Grzegorz Klima, AMLP++ document, http://amlpp.sourceforge.net/doc.html, referenced 29.04.2011

[16] NASA, Rinex 3.00 Specification, http://igscb.jpl.nasa.gov/igscb/data/format/rinex300.pdf, referenced 23.05.2011

[17] Cpluspluc, C++ reference, http://www.cplusplus.com/reference/stl/map/, referenced 29.04.2011