DELFT UNIVERSITY OF TECHNOLOGY
Faculty of Electrical Engineering
Telecommunications and Traffic Control Systems Group

Title :

A User Interface for The Magnavox MX4200D GPS Receiver

Author: E. Widjiati

Thesis report
37 pages
August 1994

Professor	:	D. van Willigen
Mentor	:	D.J. Moelker
Period	:	October 1993 - August 1994

Hereby I would like to say thank you to Dignus-Jan Moelker and Durk van Willigen for their support during my thesis work. Also thanks to all colleagues from the "Plaatsbepaling & Navigatie"-group. And special thanks to fellow students who help a lot during my stay in the group, and for their undying support.

Table of Contents

Summary iii

List of abbreviations iv

1. Introduction 1

2. Theoretical description of GPS 2
   2.1 Introduction 2
   2.2 Principle of GPS 2
   2.3 Positioning using pseudo ranges 4
      2.3.1 Time correction 4
      2.3.2 Position calculation 8
   2.4 Errors in GPS positioning measurement 10

3. Raw Data Structure of MX4200D GPS receiver 12
   3.1 Introduction 12
   3.2 Structure of Raw Satellite Data 13
   3.3 Structure of Satellite Navigation Data 15

4. Software Description 17
   4.1 Introduction 17
   4.2 Block diagram and description of the software 17
   4.3 Hatch Filtering 20
   4.4 User Screen Information 21

5. Measurement Results 23

6. Conclusions and Recommendations 27

References 28

Appendix A : Eccentric Anomaly Iteration 29

Appendix B : Satellite Position Calculation 30

Appendix C : The MX4200D Multi-port Interface 34

Appendix D : Software listing 37

Summary

The Global Positioning System is a satellite navigation system which is developed by the USAF Space Division. This system provides highly accurate position and precise time for the user. The principle of this system is the interaction between the satellites, the receiver and the control station. The subject of this report is the processing of the raw data that are received from the satellites.

In this research the Magnavox MX4200D GPS receiver is used. This GPS receiver has the multi-port as the output. One of the component of the multi-port is the raw data port. From this raw data port, the information to calculate the user or the receiver position can be obtained. This information is given in the form of the raw data which can be filtered and then used to calculate the user or the receiver position. The software to convert, filter and calculate the position are developed and described in this report.

Using the software described in this report, the user has a possibility to get the data from the receiver into the file and it can also use this data for the calculate the user position coordinates.

List of abbreviations

ADW	-	Auxiliary Data Word
CDU	-	Control and Display Unit
DGPS	-	Differential GPS
DOP	-	Dilution of Precision
ECEF	-	Earth Centred Earth Fixed
GPS	-	Global Positioning System
HOW	-	Hand-Over Word
ICD	-	Interface Control Document
ID	-	Identifier
Loran-C	-	Long Range Navigation
LS	-	Least Square method
MIAS	-	Multi-mode Integrated Approach System
MLS	-	Microwave Landing System
NMEA	-	National Maritime Electronics Association
PRN	-	Pseudo Random Number
rms	-	root mean square
RTCM	-	Radio Technical Commission for Maritime Services
SA	-	Selective Availability
SV	-	Space Vehicle
TLM	-	Telemetry Word
USAF	-	United State Air Force
UTC	-	Universal Time Coordinated

1. Introduction

The Global Positioning System is a satellite navigation system which is developed by the USAF Space Division. This system provides highly accurate position and precise time for the user. This system works using three different segments. These are the space segment, the control segment and the user segment. The space segment consists of the GPS satellites which transmit data signals to the receiver. The control segment consists of the control stations that are spread all over the world, and the user segment is the GPS receiver that is used by each user.

The signals from the GPS satellite are received by the user using a GPS receiver. One of the receiver types is the Magnavox MX4200D. This receiver has a multi port interface at the output. This interface can be used to control the receiver operation and offers raw and processed data for logging and displaying.

In this report an algorithm is described. Using this algorithm, it is possible for the user only to obtain the corrected raw data, if necessary. It can also be written into a file or further to be processed to calculate the position coordinates. This algorithm is written using a source code of Turbo Pascal (version 7.0).

The theoretical description and principle of GPS is explained in chapter 2. The information about the raw data from the GPS receiver that is used in this research is described in chapter 3. The software which is written in Turbo Pascal is explained in chapter 4. The results of the measurement are given in chapter 5. In the last chapter, the conclusions and recommendations are given.

2. Theoretical description of GPS

2.1 Introduction

The Global Positioning System (GPS) is a satellite navigation system which is developed by the Department of Defence under Air Force Management through the GPS Joint Program Office at the US Air Force (USAF) Space Division. It provides highly accurate position and velocity information in three dimensions, also the precise time, to users around the globe 24 hours a day.

This chapter has the objective to introduce the GPS and to present the algorithms required to calculate the pseudo range from time of transmission from each satellite to the user. In chapter 2.2 the principle of GPS is described. In this chapter, the GPS, which consists of three different segments, and concepts of times of transmission are defined. In chapter 2.3 the distance between the receiver and the satellites is determined from the time of transmission. Chapter 2.4 gives the classification of the error sources in a pseudo range measurement [4].

2.2 Principle of GPS

A GPS position calculation is based on the measurement of at least four distances between an observer on earth and the satellites. For each distance, a satellite is needed. Therefore, with a minimum of four satellites in view, the GPS receiver can compute its current latitude, longitude and altitude and the time in GPS system.

The GPS design consists of three segments:

1. The Space Segment

   The space segment consists of 21 satellites and three spares in six orbital planes, with four satellites in each plane. Each plane has an inclination of 55° in relation to the equator. The orbit period of each satellite is approximately 12 hours at an altitude of 10,898 nautical miles (= 20,183 km). This provides a GPS receiver with six to twelve satellites in view from any point on earth, at any particular time.

2. The Control Segment

   The control segment consists of a master control station, five monitor stations and three data uploading stations. The five monitor stations are divided evenly around the world. They are

stationed in Hawaii, Colorado Springs (master and control station), Ascension, Diego Garcia and Kwajalein. The control stations make the system work by determining the clock- and the path parameters of the satellite. These parameters are sent to the satellite each 24 hours.

3. The User Segment

The user segment consists of an equipment (receiver) which tracks and receives the signals that come from the satellites. The GPS receiver which is used in the research which is described in this report is the Magnavox MX4200D GPS receiver. The description of this GPS receiver can be found in chapter 3.

The principle of GPS is shown in Figure 2.1:

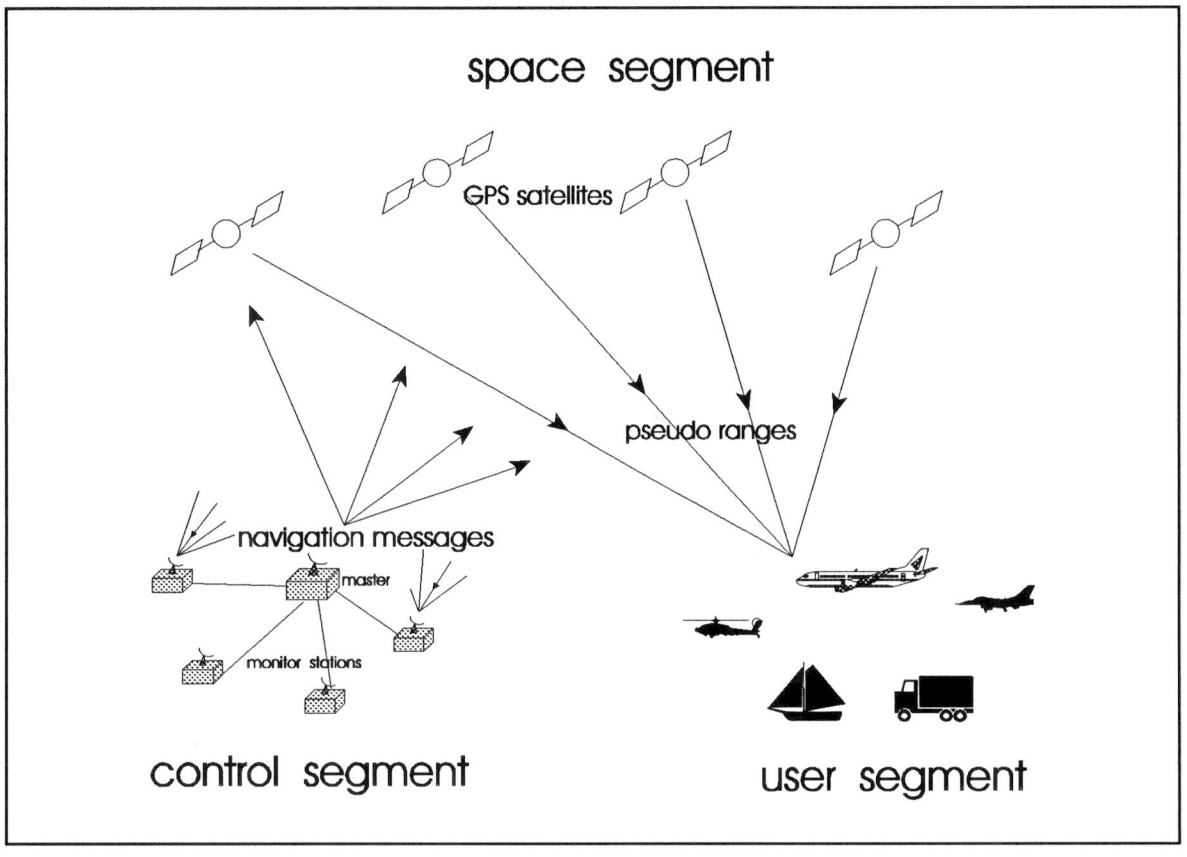


Figure 2.1 Principle of GPS

To calculate the distance from the satellite to the receiver, two independent clocks that cannot be exactly synchronized are used. These are the receiver clock and the satellite clock.[1] Because of the asynchronicity between these clocks, the traveling time of the signal from the satellite to the

[1] The relation between these two unsynchronized clocks will be described in chapter 2.3.1.

receiver contains an offset. This offset is equal to the quantity of the asynchronicity between the receiver clock and the satellite clock. This clock offset causes a constant error in the calculation of the ranges. Because of this error, this distance is called the pseudo range.

From the four different pseudo ranges, the three-dimensional position and the unknown time difference between the two asynchronous clocks can be calculated if the corresponding satellite position is known. The satellite position can be calculated using the parameters that are obtained from the signal of the accompanying satellite.[2]

The data signal that contains the satellite parameters is called the GPS navigation data message. This data message has five subframes, and it identifies the satellite (satellite issue) and provides the positioning, timing, ranging data, satellite status and the corrected ephemerides (orbit parameters) of the satellite to the users. The precise content of this message is found in chapter 3.3.

2.3 Positioning using pseudo ranges

To calculate the position of the receiver, the four different pseudo ranges with the accompanying satellite position have to be known. In the previous chapter, it is explained that to know the pseudo ranges, the clock offset that comes from the two different clock systems has to be known. The relation between these two clock systems will be discussed in chapter 2.3.1. The corrected time that is explained in chapter 2.3.1 can be used to calculate the position of the satellite and then the position of the receiver or the user. The calculation of the satellite position is given in Appendix B. The algorithm to calculate the receiver position is described in chapter 2.3.2.

2.3.1 Time correction

The GPS receiver clock system is not synchronous with the satellites clock system. The receiver clock system requires a time reference which is called the GPS time. This time reference is established by the control station. The control station designates one of the monitor station's atomic frequency standards as the GPS time.

The GPS time has a zero point defined at midnight on the night of January 5, to the morning of January 6, 1980. The longest unit used in stating the GPS-time is one week, defined as 604800 seconds. This time unit starts at the beginning of each week, i.e., Sunday 0:00 h.

The GPS time is directly related to the Universal Time Coordinated (UTC) by synchronizing the receiver clock to the UTC calibrated clock. However, the time system of the receiver or the receiver clock is not synchronous with the exact GPS system time. This is caused by a frequency

[2] The satellite position calculation is given in Appendix B.

offset and the random variation of the internal oscillator. This asynchronicity is called the clock offset.

The relation between these two clock systems in relation to the time reference is shown in the following figure:

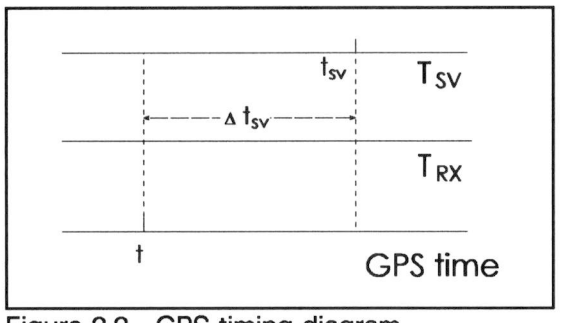


Figure 2.2 GPS timing diagram

In Figure 2.2:
T_{SV} is the time system of the satellite clock,
T_{RX} is the time system of the receiver clock which is related to a GPS time, and
GPS time is the time reference established by the control segment [3].

From Figure 2.2, the deviation of the satellite clock from the GPS time can be modeled as described in ICD-GPS-200 [1]:

$$t = t_{sv} - \Delta t_{sv} \tag{2.1}$$

where:

t	= GPS system time at the time of transmission	(seconds)
t_{sv}	= satellite clock at the time of transmission	(seconds)
Δt_{sv}	= satellite clock offset	(seconds)

In equation (2.1), the satellite clock offset is given by:

$$\Delta t_{sv} = a_{f0} + a_{f1}(t - t_{oc}) + a_{f2}(t - t_{oc})^2 \tag{2.2}$$

where:

a_{f0}, a_{f1}, and a_{f2} are the polynomial coefficients given in subframe 1 of GPS navigation data message, which can be obtained in chapter 3.3,

t_{oc} is the clock data reference time (seconds), which is also given in subframe 1 of GPS navigation data message.

In [1] it is indicated that while the coefficient's a_{f0}, a_{f1}, and a_{f2} are generated using GPS time as indicated in equation (2.2), the sensitivity of t_{sv} to t is negligible. This negligible sensitivity will allow the user to approximate t by t_{sv} in equation (2.2). It is given as:

[3] The more detailed description for this system in the used receiver is given in chapter 3.2.

$$\Delta t_{sv} = a_{f0} + a_{f1}(t_{sv} - t_{oc}) + a_{f2}(t_{sv} - t_{oc})^2 \qquad (2.3)$$

From equations (2.1) and (2.3), it follows that:

$$t = t_{sv} - \left(a_{f0} + a_{f1}(t_{sv} - t_{oc}) + a_{f2}(t_{sv} - t_{oc})^2\right) \qquad (2.4)$$

Because of trespassing at the beginning or the end of the week, if the value of $(t_{sv} - t_{oc})$ is greater than 302400 seconds, subtract 604800 seconds from t_{sv}. And if the value of $(t_{sv} - t_{oc})$ is less than −302400 seconds, add 604800 seconds to t_{sv}.

The next correction that has to be applied is the relativistic effect correction. Due to the relativistic effect, the satellite clock offset in equation (2.2) needs a correction that is described in the following equation:

$$(\Delta t_{sv})_{ck+r} = \Delta t_{sv} + \Delta t_r \qquad (2.5)$$

where Δt_r is:

$$\Delta t_r = F e \sqrt{A} \sin E_k \qquad (2.6)$$

with:

Δt_r is the relativistic correction	(seconds),
F is a constant with value $-4.442807633\ 10^{-10}$	(seconds/ meters$^{1/2}$),
e is the eccentricity of the satellite orbit	(dimensionless),
A is half the long axis of the satellite orbit	(meter),
E_k is the eccentric anomaly of the satellite orbit	(meter).

Note:
- e and A are given in subframe 2 of the GPS navigation data message (see chapter 3.3).
- E_k is iterated by means of the Picard Iteration as described in Appendix A.

Except for the correction of the clock as given in the above equations, in Annex A [2] it is given that the time of transmission has to be corrected with another correction. All of these corrections are given in Figure 2.3. In the following paragraph, the L1-L2 correction is described.

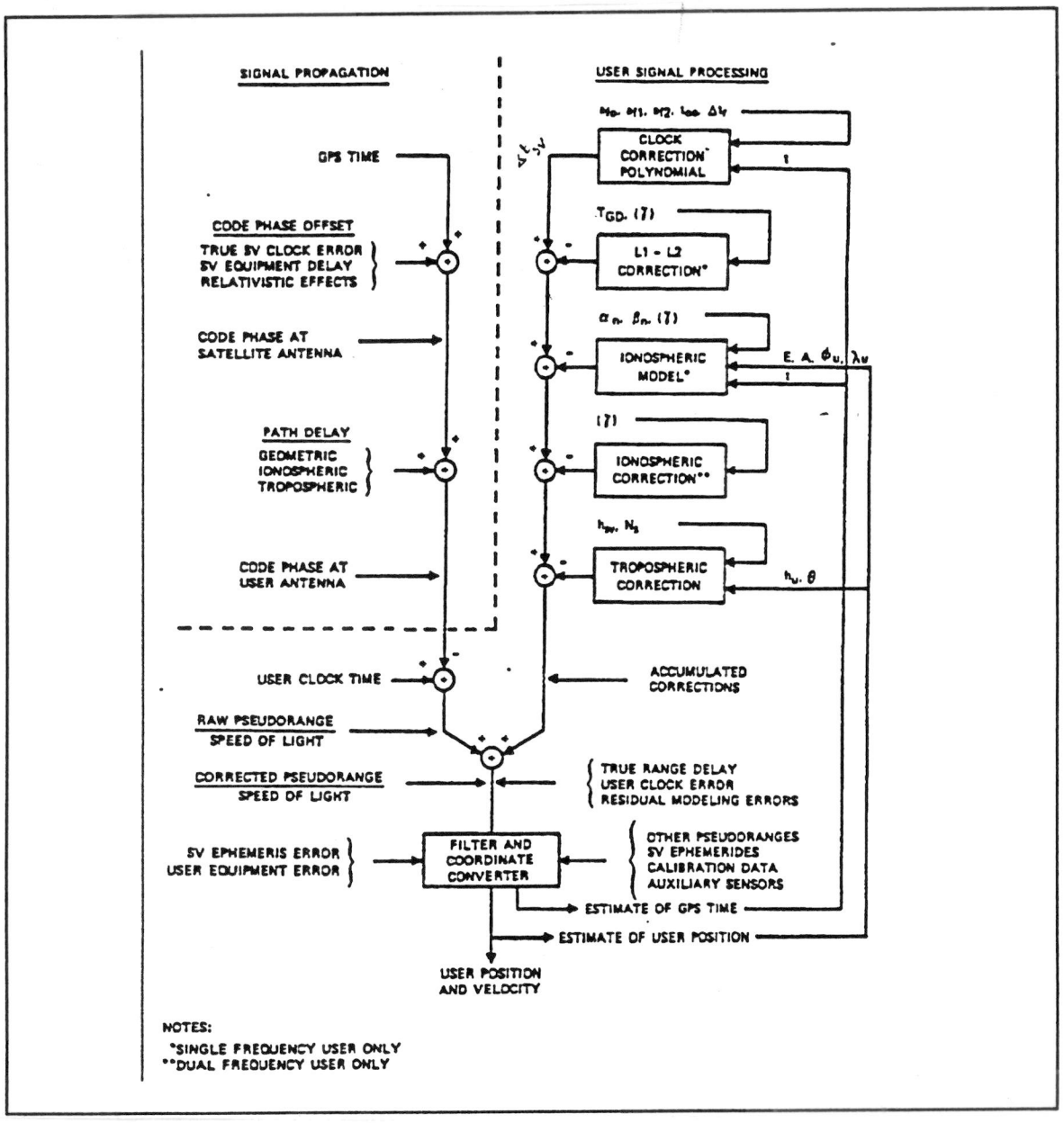


Figure 2.3 GPS positioning according to [2]

The interface between the satellite and the receiver consists of two radio frequencies L1 and L2, respectively 1575.42 MHz and 1227.60 MHz. The purpose of these two frequencies is to correct the errors as the signal propagates through the ionosphere. For the user who uses only the L1-frequency, the so-called a L1-L2 correction has to be applied. This correction is given by:

$$(\Delta t_{sv})_{L1} = \Delta t_{sv} - T_{GD} \tag{2.7}$$

where:

T_{GD} is group delay correction for L1 and L2, and transmitted in the GPS navigation data message subframe 1.

Figure 2.3 shows that the other corrections which are included in the correction of the clock are the ionospheric and the tropospheric corrections. In appendix 6 in Annex A [2], the mathematical models for ionospheric and tropospheric corrections are listed. In the calculation algorithm for the corrected time of transmission, that is described in this report, these corrections are not included.

2.3.2 Position calculation

From chapter 2.2, it is known that to calculate the position of the receiver, at least four pseudo ranges and the corresponding satellite position are needed. This position calculation is given in this chapter. The satellite position can be calculated using coefficients that are received from the GPS navigation data message. The content of this navigation data message is briefly described in chapter 3.3 and can also be found in ICD-GPS-200 [1]. The calculation of the satellite position is given in appendix B.

From the four pseudo ranges and the corresponding satellite positions, the position of the user can be calculated. The mathematical model is given by Brouwer [3]:

$$R^j = PR^j + \Delta R = \sqrt{(x - x^j)^2 + (y - y^j)^2 + (z - z^j)^2} \tag{2.8}$$

where:

R^j is the true range from the receiver to the satellite,
PR^j is the measured pseudo range,
$\Delta R = c\ \Delta T$, is the error in the pseudo range,
c is the speed of light (from [1] it is given that $c = 2.99792458\ 10^8$ m/s),
ΔT is the clock offset at the GPS time,
x, y, z are the coordinates of the user,
x^j, y^j, z^j are the coordinates of the j^{th} satellite.

In equation (2.8), there are four unknowns x, y, z and ΔT. To solve this equation, it has to be linearized as follows:

$$R^j\, dR^j = (x - x^j)\, dx + (y - y^j)\, dy + (z - z^j)\, dz$$

or

$$dR^j = \frac{(x - x^j)}{R^j}\, dx + \frac{(y - y^j)}{R^j}\, dy + \frac{(z - z^j)}{R^j}\, dz \tag{2.9}$$

A measurement from a satellite j can be given as:

$$R^j = R_0^j + dR^j \tag{2.10}$$

where:

R_0^j is the geometrical distance between the receiver and the satellite, and
dR^j is the error of the measurement.

Substitution of dR^j from equation (2.9) and R^j from equation (2.8) in equation (2.10), gives:

$$R_0^j + \frac{x_0 - x^j}{R_0^j} dx + \frac{y_0 - y^j}{R_0^j} dy + \frac{z_0 - z^j}{R_0^j} dz = PR^j + \Delta R \tag{2.11}$$

This can be written as a general matrix equation as follows:

$$A \cdot \underline{u} = \underline{v} \tag{2.12}$$

where:

$$A = \begin{bmatrix} \frac{x_0 - x^1}{R_0^1} & \frac{y_0 - y^1}{R_0^1} & \frac{z_0 - z^1}{R_0^1} & 1 \\ \frac{x_0 - x^2}{R_0^2} & \frac{y_0 - y^2}{R_0^2} & \frac{z_0 - z^2}{R_0^2} & 1 \\ \frac{x_0 - x^3}{R_0^3} & \frac{y_0 - y^3}{R_0^3} & \frac{z_0 - z^3}{R_0^3} & 1 \\ \cdots & \cdots & \cdots & \cdots \\ \frac{x_0 - x^n}{R_0^n} & \frac{y_0 - y^n}{R_0^n} & \frac{z_0 - z^n}{R_0^n} & 1 \end{bmatrix} \quad ; \quad \underline{v} = \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \\ \Delta R \end{bmatrix} \quad ; \quad \underline{u} = \begin{bmatrix} dx \\ dy \\ dz \\ dR \end{bmatrix}$$

$\underline{v} = PR^j - R_0^j$ is the difference between the pseudo range measurement and actual distance, and
n is the number of satellites.

If the number of satellites is equal to the number of unknowns, $\underline{u}$ can be solved as:

$$\underline{u} = A^{-1} \cdot \underline{v} \tag{2.13}$$

If the number of satellites is more than the number of unknowns, $\underline{u}$ can be solved using the Least Squares (LS) method. Using this method, the best estimate of the solution can be calculated. The formula of Least Square method then follows:

$$\underline{u} = (A^T A)^{-1} A^T \cdot \underline{v} \tag{2.14}$$

with A^T is the transpose of the matrix A.

2.4 Errors in GPS positioning measurement

The GPS signal that is received by the user from the satellite, contains many errors that are caused by several sources, such as the ionosphere, troposphere, clock errors and multipath errors. All these errors cause inaccuracy in the position calculation.

In the article written by Braasch [4], a GPS signal model is presented. This signal model can be used for certain simulation purposes. For a simulation using this signal model, error sources are generated into one model that computes the observed pseudo range for each satellite in view and that determine the perturbed user position.

The error source which is described in [4], is given as:

$$\varepsilon = \varepsilon_{ck,rcvr} + \varepsilon_{ck,sv} + \varepsilon_{tropo} + \varepsilon_{iono} + \varepsilon_{URE} + \varepsilon_{mp} + \varepsilon_{noise} + \varepsilon_{SA} \tag{2.15}$$

with:

$\varepsilon_{ck,rcvr}$	is receiver clock offset from GPS time,
$\varepsilon_{ck.sv}$	is satellite clock offset from GPS time,
ε_{tropo}	is propagation delay error caused by the troposphere,
ε_{iono}	is propagation delay error caused by the ionosphere,
ε_{URE}	is user range error,
ε_{mp}	is multipath errors,
ε_{noise}	is receiver measurement noise (clock noise and diffuse multipath), and
ε_{SA}	is error by Selective Availability.

Note:

- The receiver clock offset is the clock offset as estimated from the observed pseudo ranges.
- The satellite clock offset is the clock offset as decoded from the satellite navigation data message.
- The user range error contains the satellite ephemeris errors projected onto the line-of-sight direction and the residual satellite clock errors.

- Selective Availability is a standard positioning service accuracy which is developed by the US Department of Defense for security reasons.

These error sources can be computed for the observed pseudo range for each satellite in view and determines the perturbed user position. From SS-GPS-300C [12], these errors are given as follows:

Table 2.1 Error sources in GPS signal

	errors (ft)
The satellite clock offset	10
Troposphere delay	6
Ionospheric delay	27
Multipath	10
Receiver noise	30
Selective Availability	90

3. Raw Data Structure of MX4200D GPS receiver

3.1 Introduction

The Magnavox MX4200D GPS receiver is the receiver used in the research described in this report. This Magnavox GPS receiver has several operational capabilities. These capabilities are controlled by the Magnavox developed Control and Display Unit (CDU) program. The CDU program is used to perform the control and to operate as the display function. The description of each function in this CDU can be found in User's Guide of this GPS receiver [5].

The output of this GPS receiver is a multi port interface that offers the user the possibility to control the receiver to use particular messages. It can also be used to control an interface to external equipment and to control the output of raw and processed data for logging and display. For one type of receiver, i.e. the MX4200D, differential corrections from a GPS reference station can be received and processed.

The multi port interface consists of four serial bidirectional data ports. Each data port has a specific function. The first port is a control port that can be used to initialize, monitor and control the MX4200D. The second port is an equipment port that can be used to allow the MX4200D to communicate with external equipment. The third port is a raw data port that is used to give an output of raw and processed data from the MX4200D in a Magnavox defined format. This port gives information such as raw satellite measurements, position, velocity, ephemeris and almanac (from the GPS navigation data). The fourth port is a RTCM port that is only used by the MX4200D (not by the MX4200 type). This port is used to receive the differential corrections from a GPS reference station.

For the research described in this report, the raw data port and some control port types are used. The content of the complete raw data port codes and some of the control port codes is given in appendix C. In this chapter, two message types of raw data will be described. These are the raw satellite data and the satellite navigation data. In chapter 3.2, the content of the raw satellite data will be described. This data is needed to calculate the precise time of transmission that comes out of the receiver. The structure of the satellite navigation data is given in chapter 3.3. In this chapter, the parameters to calculate the satellite coordinates as in Appendix B, and the satellite clock error are specified.

3.2 Structure of Raw Satellite Data

The raw satellite data from the MX4200D GPS receiver is given in two forms, the record type 1 uncompressed raw satellite data and the record type 2 compressed raw satellite data. These two record data types contain the same information. The difference between these two forms is the difference in the control port of each record type.

The record type 1 of the raw satellite data type has the control port with record type $PMVXG, 024. This raw satellite data type contains the information that comes from the satellite as listed in Table 3-1. The contents of this data port (data port with record type 1 or the decoding of record type 2) are also given in [6].

The raw satellite data from record type 2 has the control port record type $PMVXG, 027. This raw satellite data type has five ASCII character with header '2' and followed by 71 data characters and one checksum character. Each data character contains seven bits of binary data that have been mapped into a 'printable' subset of the extended ASCII character set. An algorithm to decode the compressed form to the uncompressed form is given in [6], and also listed in the source code of Turbo Pascal, as will be described in chapter 4.

Table 3-1 Raw Satellite Data Record

Byte location	Identifier	Description	Range
1-4	ID	Record ID	1
6-7	CHNL	Receiver channel number	1-6
9-10	PRN	Satellite PRN	1-32
12-20	USER_MS	User time of measurement (ms)	0-604799999
22-30	CHNL_MS	Channel time of measurement (ms)	0-604799999
32-41	PHI	Integrated carrier phase in L1 wavelengths	--
43-48	CODE	Raw code offset in L1 wavelengths	--
50-53	PHI-FRAC	Integrated carrier phase, fractional portion	-128 - 127
55-58	CR	Costas ratio	(-128 - 127)
60-63	SNR	Signal-to-noise ratio at 1 Hz bandwidth, in dB-Hz	25 - 53
65		Half-cycle phase ambiguity indicator	(- , + , ?)

The important elements from Table 3.1 which are used here are:

- **USER_MS** :

  This is the time of the measurement based on the local receiver clock. This time is given in milliseconds.

- **CHNL_MS** :

  This is the time of transmission based on the satellite clock. This time is given in milliseconds.

- **PHI** :

  This is the integrated carrier phase that is given in whole cycles.

- **PHI_FRAC** :

  This is the integrated carrier phase which is given in fractional portion.

- **CODE** :

  This is the code phase offset.

The relation between these definitions is illustrated in Figure 3.1:

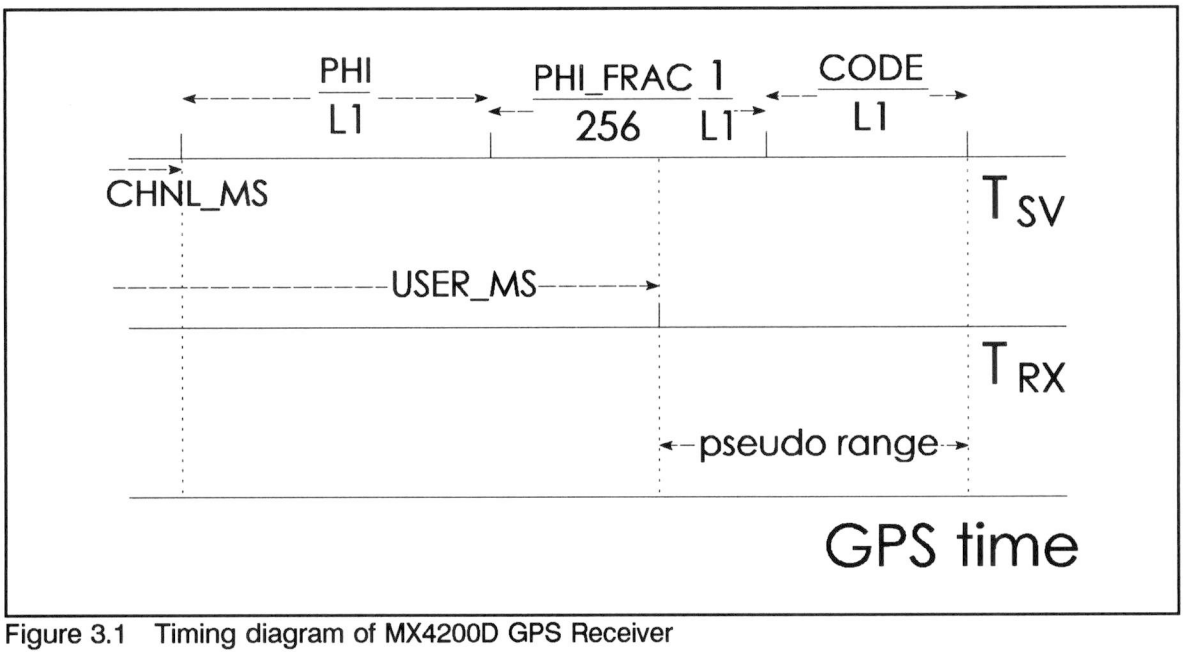


Figure 3.1 Timing diagram of MX4200D GPS Receiver

Note:

- L1 is the frequency of the signal which is sent by the satellite.

There are two terms mentioned in the above definitions. These are the integrated carrier phase and the code phase. The phase measurement is the measurement of the phase comparison between the carrier which is sent by the satellite and the received signals in the receiver.

The code phase offset measurements are usually referred to as pseudo range measurements. The carrier phase tracking measurements have two forms, the integrated Doppler

count and the relative phase of the received carrier phase and the receiver clock phase at a particular epoch.

Using the above definitions, in lit [6] is given that the user time or the time of measurement is:

$$User\_Time = \frac{USER\_MS}{1000} \quad [seconds] \tag{3.1}$$

And the time of transmission for the satellite is:

$$Channel\_Time = \frac{CHNL\_MS}{1000} + \frac{PHI + \frac{PHI\_FRAC}{256}}{L1\,FREQ} + \frac{CODE}{L1\,FREQ} \quad [seconds] \tag{3.2}$$

From the two equations above, the pseudo range measurement is:

$$Pseudo\_Range = (User\_Time - Channel\_Time) * c \quad [meters] \tag{3.3}$$

with: c is the speed of light (= 299 792 458 m/s).

3.3 Structure of Satellite Navigation Data

The satellite navigation data contains the information of the satellite at the moment of measurement. This includes the position of the satellite and the offset of the satellite clock in relation to the GPS time. All these parameters are given in the GPS satellite navigation data which is transmitted by every GPS satellite. This data is illustrated in Figure 3.2.

The satellite navigation data consist of 1500 bits which is transmitted at 50 bits per second. These bits are divided into five subframes of 300 bits each. Each subframe contains ten words. The first two words are a TLM (Telemetry Word) and a HOW (Hand Over Word). The TLM message contains information needed by the authorized user. The format and content of HOW can be found in ICD-GPS-200 [1].

The GPS satellite navigation data contains five subframes followed by frame numbers.
These are:

- Subframe 1 - clock offset correction

  It tells the receiver how much the satellite clock differs from GPS time.

- Subframes 2 and 3 - ephemeris predictions

  Both provide precise description of the satellite orbit to obtain a position. This remains accurate for about 4 hours.

- Subframe 4 - correction data

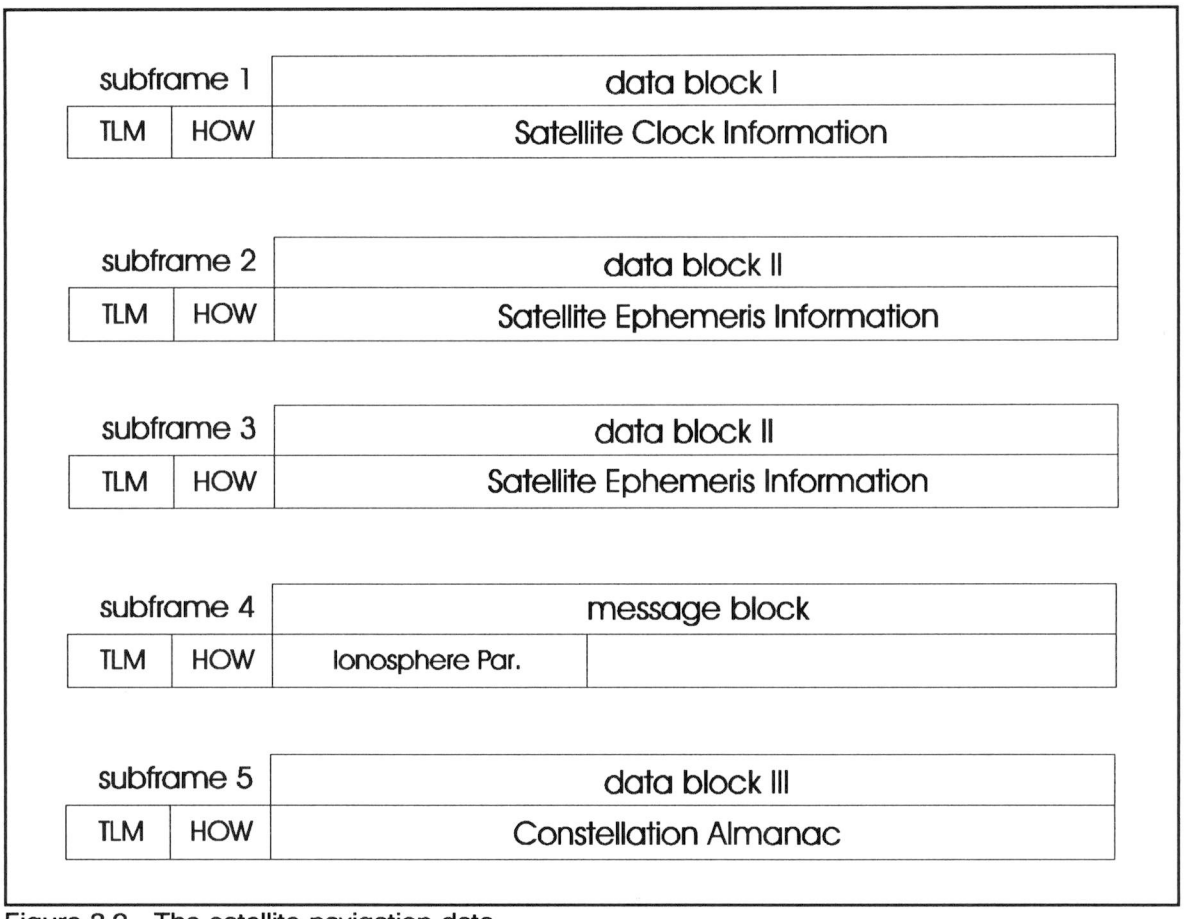


Figure 3.2 The satellite navigation data

It compensates for the delay of radio waves in the atmosphere, and some data for the conversion GPS time to Universal Time Coordinated (UTC).

- Subframe 5 - almanac information

  It gives the GPS receiver the approximate location of all operating satellites. Each 30-second message provides a subframe of the almanac.

The parameters of subframe 1 of the GPS satellite navigation data is used to calculate the satellite clock errors as described in chapter 2.3.1. These parameters are for example week number, Issue of Data Clock (IODC), estimated group delay differential (T_{GD}) and parameters of the clock corrections (t_{oc}, a_{f2}, a_{f1} and a_{f0}). The week number represents the number of the current GPS week at the start of the data set transmission interval. The GPS week number increases at each end/start of each week. The Issue of Data Clock gives the issue number of the data set. It provides the user with a convenient means of detecting any change in the correction parameters.

The parameters of subframes 2 and 3 are used to calculate the position of the satellite at the time of measurement. The content of these subframes and the equations to calculate the satellite coordinates are given in Appendix B.

4. Software Description

4.1 Introduction

In the previous chapter, the algorithm to calculate the position of the user is given. This algorithm is implemented to a program with the source code of Turbo Pascal (version 7.0). The Turbo Pascal program is divided into several units. A unit is a precompiled collection of Turbo Pascal program, which can also be used in other programs. A unit can store anything that can be found in a normal Turbo Pascal program, but it is not a complete program. It cannot be run separately. Instead, a main program has to use the unit's features, as if those features are declared directly in the program.

In this chapter, the program that consists of several units to calculate the position of the receiver using the data from the receiver is described. The block diagram, the relations between the units and the description of the software are explained in chapter 4.2. In chapter 4.3 the filter which is used will be described. This is the so-called Hatch filter. The choices that the user can make using this software are illustrated in chapter 4.4.

4.2 Block diagram and description of the software

The research using the Magnavox MX4200D has been done before in two other projects by the Telecommunication and Traffic Control System group in the Faculty of Electrical Engineering, Delft University of Technology. These projects are Eurofix and MIAS.

The concept of Eurofix is a combination of Loran-C and GPS in which the Differential GPS corrections are transmitted to the user by modulating the Loran-C signal. In this project, the GPS receiver has a function as the reference station or mobile receiver.

The concept of MIAS is a hybrid approach system, based on the positioning system MLS and DGPS. In this project, the GPS receiver provides a DGPS correction data. This correction data is transmitted to the users through the MLS ADW data channel, where the DGPS data is used in another GPS receiver (mobile receiver).

In the Eurofix research done by L.J.Beekhuis and reported in [8], the GPS receiver that is used is also the MX4200D type. The diagram to obtain the differential correction from the GPS receiver are explained in the software documentation of this research (Beekhuis [10]). General parts of the

algorithm in that diagram are also used in the research that is described in this report. These are step1.exe and step2R.exe. The step1.exe is used to convert the raw satellite data from the compressed format to the uncompressed format. And the step2R.exe is used to calculate the satellite positions and to obtain the range errors, if the reference station position is known. Part of step1.exe and the algorithm to calculate the satellite positions from step2R.exe is used here. In the diagram from [10], the Least Square method is given in the Matlab format, and here it will be translated in Turbo Pascal format.

The diagram which describes the position calculation algorithm explained in the following paragraphs is given in Figure 4.1.

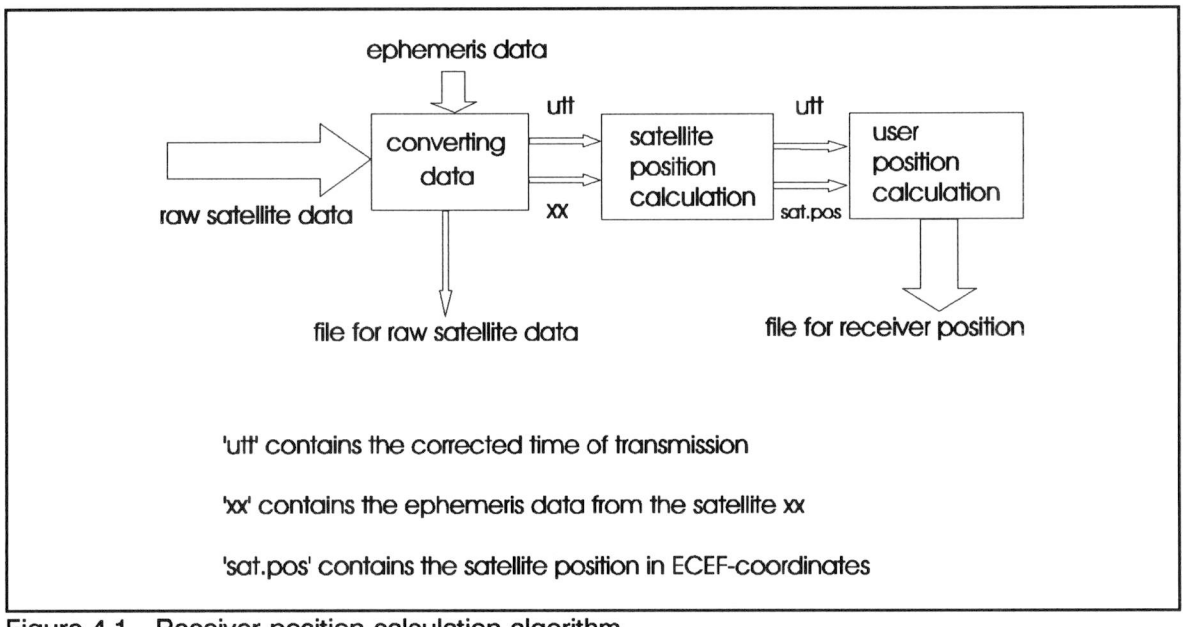


Figure 4.1 Receiver position calculation algorithm

The first step in this algorithm is the conversion of the compressed raw satellite data from the raw data port of the receiver. The compressed raw satellite data is received from the satellite receiver with record type 2. This data is converted from the compressed format to the uncompressed format and then kept in the memory and also stored in files. One of the important files contains the corrected time of transmission and has an *.utt extension. Other important files contain the satellite ephemeris data and have their own satellite number in the extension. In the conversion, the Hatch filter is also used to filter the raw code offset from the receiver. The principle of this filter is explained in chapter 4.3.

The second step of this algorithm is the calculation of the satellite position. This calculation can only be done if the satellite ephemeris data is received from the satellite. This satellite ephemeris data is sent by the receiver with record types 200 through 203. The corrected time of transmission

and the satellite ephemeris data are then used to calculate the satellite position. The equations to calculate the satellite position are listed in Appendix B.

The user position calculation using the Least Squares method is done as the last step. This method is described in chapter 2.3.2. The corrected times from the first step and the calculated satellite position from the second step are the given parameters which are needed for this calculation.

From the algorithm described in Figure 4.1, a main Turbo Pascal program with its units is illustrated in Figure 4.2.

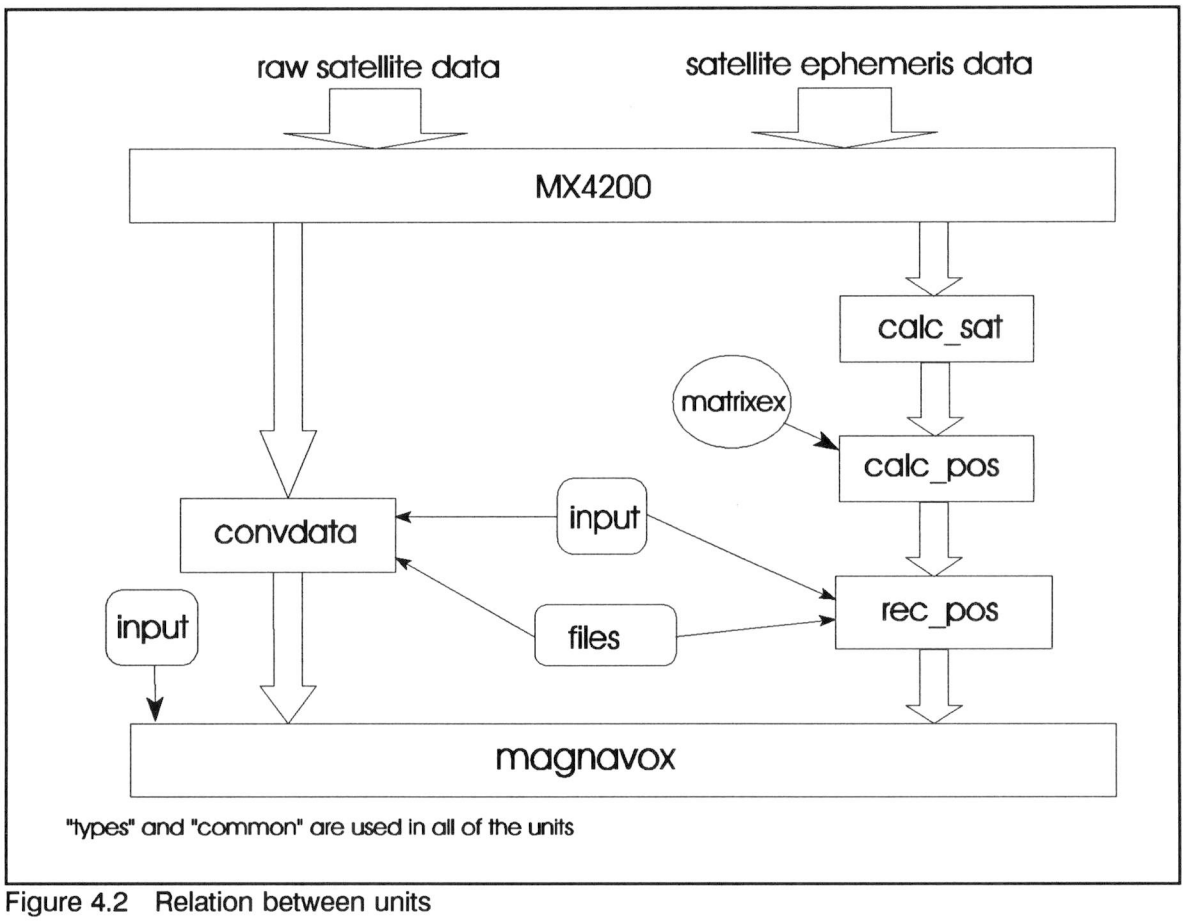


Figure 4.2 Relation between units

From Figure 4.2 it can be seen that the 'magnavox' is the main program of all the units. In this program, two important units are written. They are 'convdata' and 'rec_pos'. In 'convdata', the compressed raw satellite data is converted into the uncompressed form. And in 'rec_pos', the user position is calculated.

In the main program 'magnavox', there are also two units which contain the definition of all types and common procedures that are used in all units. These units are 'types' and 'common'. In 'types', the type definitions which are used in all units are given. The 'common' contains the functions and the procedures that are used to communicate with the user. These are for example, the interface to the screen and the interface with the keyboard.

In the following subsections the two important units of the main program will be explained.

- **convdata**

In this unit the conversion of the compressed raw satellite data form to the uncompressed form is performed using two other units:

- '**MX4200**", which converts the compressed raw satellite data form to the uncompressed form. The listing of the program to convert this data is given in [5, Appendix D]. In this literature, the program is written in the C-program language.
- '**files**', in which the result is written to the files.

- **rec_pos**

This unit uses other units, such as:

- '**calc_sat**', which calculates the position of the satellite. This unit has as input the result of 'MX4200'.
- '**calc_pos**' calculates the position of the user using the Least Squares method. In this unit, matrix calculation is performed using 'matrixex' unit.

4.3 Hatch Filtering

The principle of Hatch filtering is the extrapolation of the old code measurement and the current code measurement using the Doppler count or 'change of range' measurement [11]. For the current pseudo range determination, one pure code measurement is available and many 'old' code measurements which are extrapolated using the Doppler count. By averaging all these 'Code' measurements, the noise of the single code measurement is reduced. In this way Hatch filtering over two measurements becomes:

$$\overline{PR_j^2} = PR_j + \left(PR_{j-1} + DR_j\right) \tag{4.1}$$

with:

PR_j^2 is the Hatch filtered pseudo range over two measurements
PR_j is the measured code pseudo range at time j
DR_j is the Doppler count measurement at time j

The general equation for Hatch filtering over N measurements becomes:

$$\overline{PR_j^N} = \frac{1}{N}\left(PR_j + \sum_{k=j-N+1}^{j-1}\left(PR_k + \sum_{p=k+1}^{j} DR_p\right)\right) \tag{4.2}$$

with PR_j^N is the Hatch filtered pseudo range over N measurements.

The Hatch filter is used in the diagram in Figure 4.2 to filter the code carrier offset from the Magnavox MX4200D GPS receiver. From the time definitions given in chapter 3.2, the non-recursive Hatch filtering for the MX4200D becomes [8]:

$$(\text{Channel\_Time})_j = \frac{\text{CHNL\_MS}_j}{1000} + \frac{\text{PHI}_j + \frac{\text{PHI\_FRAC}_j}{256}}{\text{L1FREQ}} + \frac{1}{N}\frac{1}{\text{L1FREQ}}\sum_{k=I-N+1}^{I} \text{CODE}_k \tag{4.3}$$

4.4 User Screen Information

The receiver position algorithm is given in Figure 4.1. The more detailed form of this figure is reillustrated in Figure 4.3.
In this figure, it can be seen that the raw satellite data from the GPS receiver is processed in the "converting data" block. From this block, the user has the choices to give the name of the output-file. Here, it is given as a downward arrow.

The output files from this part are:

- *.utt - the file which contains the corrected time of transmission.
- *.xx - the file which contains the satellite ephemeris data. The xx-code corresponds to the satellite number.
- *.pos - the file which contains the user's position that the MX4200D gives. This data is taken from the raw data with record type 8.
- *.sat - the file which contains the satellite geometry. This data is obtained from the raw data with record type 411 until 416.

After this block, the satellite positions are calculated. The result of this calculation is used as the input for the next block. The next block is the user position calculation. From this calculation, the user can get the file which contains the coordinates of the receiver in ECEF-coordinates. The output of this calculation is a file with the extension *.rec.

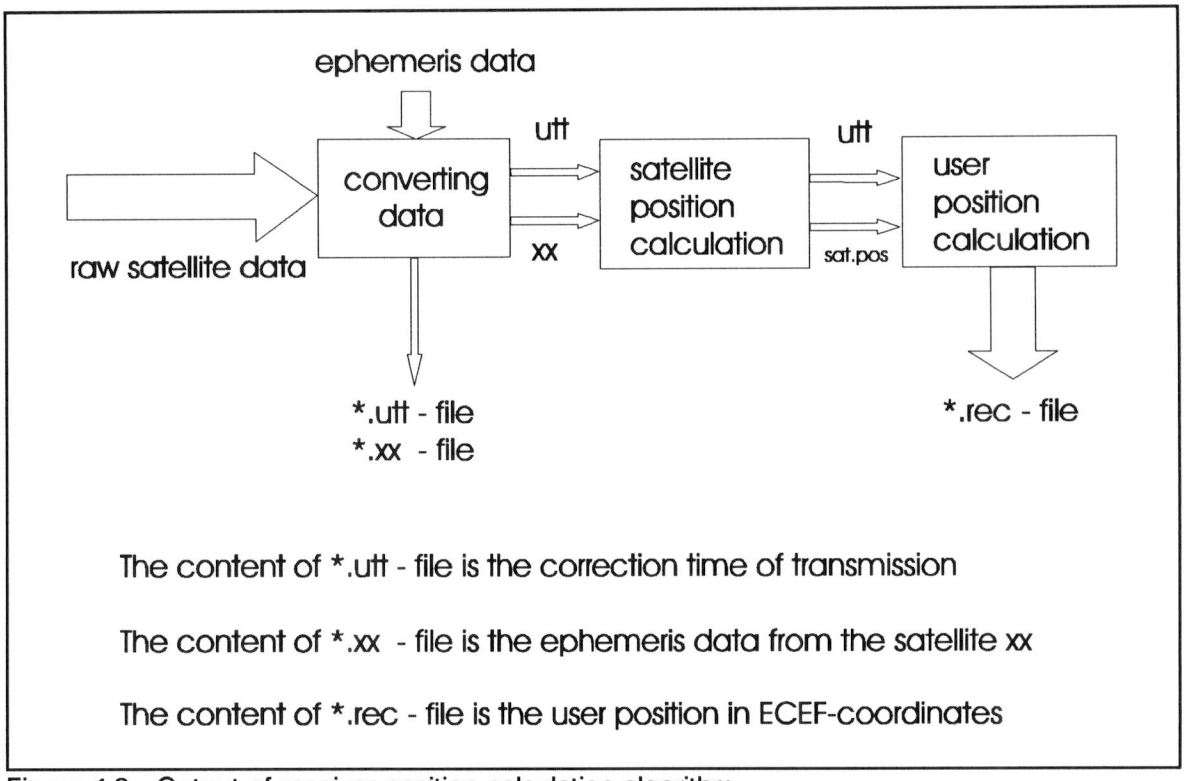


Figure 4.3 Output of receiver position calculation algorithm

5. Measurement Results

Using the software 'magnavox' (described in chapter 4 and listed in appendix D), the user can obtain the output from the MX4200D in the form of a file. The output can also be stored in the form of variables as defined in one of the units ('types.pas'). These variables are then further used to calculate the user position. The input data file of this program is a file with extension '*.dat'.

The menu that the user can get at the screen from the 'magnavox.exe' is:

```
Program for MX 4200D GPS Receiver

1 : Convert the raw data to files

2 : User position calculation

3 : exit
```

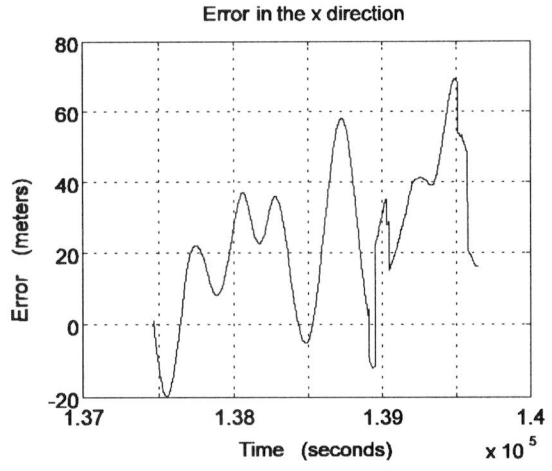
The first menu item, 'Convert the raw data to files' means that the user has a choice which data would be put into the file with extension '*.utt'. There are four choices possible. These are:
- The corrected time of transmission
- The pseudo range
- The carrier phase
- The code phase

During the conversion the user can see on the screen which satellite data are used in the calculation.

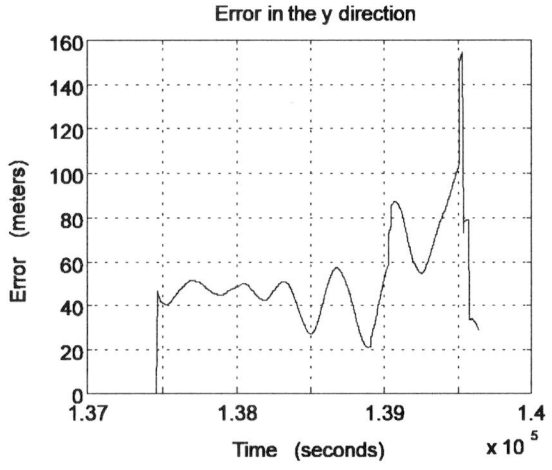
After the 'User position calculation' menu item has been executed, the user has a file with the extension '*.rec' which contains the position of the user in ECEF-coordinate system. During the running of this part, the user can see which satellite that is being tracked, and also the calculated position coordinates.

Using the 'User position calculation' part, we can calculate the user position. From the data that are measured at 18 July 1994 at Faculty of Electrical Engineering, Delft University of Technology, the x, y, and z coordinates are computed. The errors from these calculations are given in the figures below.

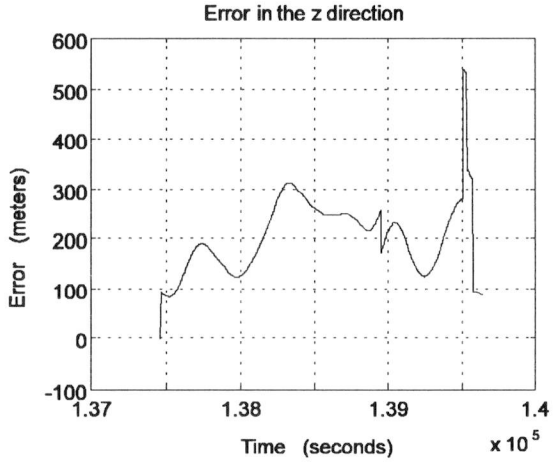
The x-coordinates in ECEF:



The y-coordinates in ECEF:



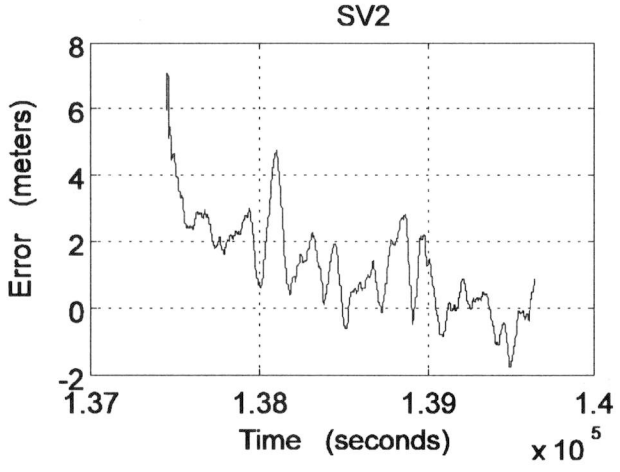
The z-coordinates in ECEF:



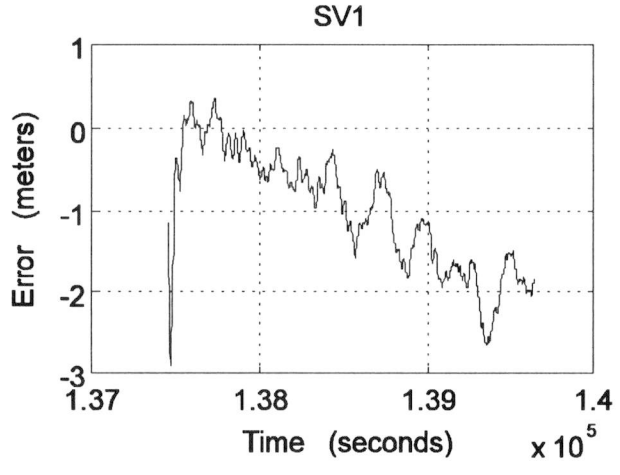
- From the above figures, the largest part of the errors is caused by SA. This error is introduced by the US Department of Defence for security reasons.
- The jump in the calculation of the x, y, and z coordinates occured if the data from a particular satellite dropped out or if the other satellite data is coming in.

From the error sources given in chapter 2.4, the multipath error, ionosphere errors from the pseudo range and the carrier phase measurement, and the noise from the pseudo range measurement can be obtained from the difference between the measurement of the pseudo range and the carrier phase. The plot of this difference for several satellites is given in the following figures.

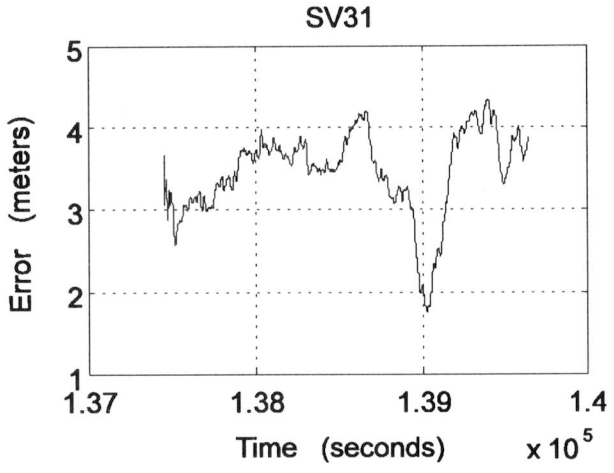
The plot of the pseudo range - carrier phase from SV2.



The plot of the pseudo range - carrier phase from SV1.



The plot of the pseudo range - carrier phase from SV31.



6. Conclusions and Recommendations

Conclusions

- From the results of the previous chapter, we can conclude that the position calculation is influenced by the ephemeris data from the satellite. In the case that another satellite is used as replacement for example, the ephemeris data received will be different, and this is the satellite jump. This satellite jump causes a difference in the position calculation.

- The pascal program used in this research works with units. This makes it possible to add other units for different purposes.

Recommendations

- This user interface can be used for further research in this group.

- In the further research, the differential GPS can be applied to obtain a better accuracy than the GPS alone.

- To get a better accuracy in the position calculation, it may be better to use filter in other part(s) of the software.

References

[1] ICD-GPS-200 Interface Control Document, NAVSTAR GPS, 3 July 1991, Arinc Research Corporation, Navstar GPS Space Segment, Navigation User Interface

[2] Annex A to STANAG 4294, Standardization Agreement, 'NAVSTAR Global Positioning System (GPS), System Characteristics - Preliminary Draft', Draft issue L.
MAS NATO, 1 August 1990.

[3] Brouwer, F.J.J., GPS, Navigatie en Geodetische Puntsbepaling met het Global Positioning System, Delft Universitaire Pers, 1989.

[4] Michael S. Braasch, A Signal Model for GPS, Navigation: Journal of the Institute of Navigation, Vol. 37, No. 4, Winter 1990-91, pp. 363-377.

[5] MX 4200 GPS Receiver, MX 4200 PC Controller User's Guide, Magnavox Electronics System Company 1991, West Coast Division, Torrance, California, R-7144A, October 1991, No. 60226.

[6] MX 4200 GPS Receiver Programmer's Guide, Magnavox Electronics System Company 1992, West Coast Division, Torrance, California, R-7145A, February 1992, No. 60222.

[7] MX 4200 GPS Receiver, Installation and Service Manual, Magnavox Electronics System Company 1991, West Coast Division, Torrance, California, R-7122B, October 1991, No. 60232.

[8] Beekhuis, L.J., The Asynchronous Correction Concept for High Precision DGPS in Eurofix, thesis report, Delft University of Technology, Faculty of Electrical Engineering, June 1993.

[9] Meijer, R.C., Development and testing of the airborne part of the MLS Integrated Approach System (MIAS), thesis report, Delft University of Technology, Faculty of Electrical Engineering, March 1993.

[10] Beekhuis, L.J., Software documentation of "The asynchronous correction concept for high precision DGPS in Eurofix".

[11] Hatch R, The Synergism of GPS Code and Carrier Measurements, Proceedings of the Third International Geodetic Symposium on Satellite Doppler Positioning, Las Cruces, NM, February 1982, pp. 1213-1232.

[12] GPS System Specification, SS-GPS-300C, USAF.

Appendix A Eccentric Anomaly Iteration

The orbit of the satellite is illustrated in the ellipse form. This ellipse form has an eccentric anomaly that can be described according to a Keppler equation:

$$M_k = E_k - e \sin E_k \tag{A.1}$$

Note: The variable e, the eccentricity of the Kepler orbit is given in the GPS navigation data subframe 2 and 3 (ephemeris data).

Equation (A.1) can be solved using the Picard Iteration as follows:

$$(E_k)_{n+1} = M_k + e \sin(E_k)_n \tag{A.2}$$

The starting value of E_k is M_k from equation (A.1).

Using Newton Raphson, the quadratic convergence occurs if:

$$(E_k)_{n+1} = (E_k)_n - \frac{f(E_k)_n}{f'(E_k)_n} \tag{A.3}$$

In above equation, the function of f is:

$$f(E_k) = M_k - E_k + e \sin(E_k) \tag{A.4}$$

and

$$f'(E_k) = -1 + e \cos(E_k) \tag{A.5}$$

This iteration is finished when the number of iteration is more than the maximal allowed number of iterations or

$$|(E_k)_{n+1} - (E_k)_n| < \epsilon$$

Note: In the research for this report, the accuracy for Newton Raphson is 10^{-12} and the maximal number of iteration is 10.

Appendix B Satellite Position Calculation

The GPS navigation data subframes 2 and 3 contain the GPS ephemeris data. This data are needed to calculate the position of the satellite. In [1, Table 20-III] the contents of this data is listed. This data is given also in the following paragraph.

The broadcasted satellite ephemeris contains:

M_0	=	the mean anomaly at reference time (semi-circles)
Δn	=	the mean motion difference from computed value (semi-circles/seconds)
e	=	eccentricity (dimensionless)
$(A)^{1/2}$	=	the square root of the semi-major axis ($meters^{1/2}$)
Ω_0	=	the longitude of the ascending node of orbit plane at weekly epoch (semi–circles)
i_0	=	the inclination angle at reference time (semi-circles)
ω	=	the argument of perigee (semi-circles)
Ω	=	the rate of Right Ascension (semi-circles/seconds)
IDOT	=	the rate of Inclination Angle (semi-circles/seconds)
C_{uc}	=	the amplitude of the cosine harmonic correction term to the argument of latitude (radians)
C_{us}	=	the amplitude of the sine harmonic correction term to the argument of latitude (radians)
C_{rc}	=	amplitude of the cosine harmonic correction term to the orbit radius (meters)
C_{rs}	=	amplitude of the sine harmonic correction term to the orbit radius (meters)
C_{ic}	=	amplitude of the cosine harmonic correction term to the angle of inclination (radians)
C_{is}	=	amplitude of the sine harmonic correction term to the angle of inclination (radians)
t_{oe}	=	reference time ephemeris (seconds)
IODE	=	issue of data (ephemeris)

The user can compute the earth fixed coordinates of the satellite antennas phase center using variations of the equations shown in [1, Table 20-IV]. These equations are also listed in the following paragraph.

The WGS-84 value of the Earth's Universal Gravitational Constant is:

$$\mu = 3.986005 \times 10^{14} \frac{meter^3}{sec^2} \tag{B.1}$$

The WGS-84 value of the Earth's rotation rate is:

$$\dot{\Omega}_e = 7.2921151467 \times 10^{-5} \frac{rad}{sec} \tag{B.2}$$

The semi-Major Axis is:

$$A = \left(\sqrt{A}\right)^2 \tag{B.3}$$

The Computed mean motion is:

$$n_0 = \sqrt{\frac{\mu}{A^3}} \tag{B.4}$$

If t is the time from Ephemeris Reference Epoch then:

$$t_k = t_{tx_{GPS}} - t_{oe} \tag{B.5}$$

The Corrected mean motion is:

$$n = n_0 + \Delta n \tag{B.6}$$

The Mean anomaly is:

$$M_k = M_0 + n \cdot t_k \tag{B.7}$$

To get the Eccentric anomaly we solve iteratively Kepler's equation for Eccentric anomaly:

$$M_k = E_k - e \cdot \sin(E_k) \tag{B.8}$$

The True Anomaly is:

$$\nu_k = \operatorname{atan}\left(\frac{\sqrt{1-e^2} \cdot \sin(E_k)}{\cos(E_k) - e}\right) \tag{B.9}$$

The Argument of latitude is:

$$\Phi_k = \nu_k + \omega \tag{B.10}$$

The Argument of Latitude correction for second harmonic perturbations is:

$$\delta u_k = C_{us} \cdot \sin(2\Phi_k) + C_{uc} \cdot \cos(2\Phi_k) \tag{B.11}$$

The Radius correction for second harmonic perturbations is:

$$\delta r_k = C_{rc} \cdot \cos(2\Phi_k) + C_{rs} \cdot \sin(2\Phi_k) \tag{B.12}$$

The correction to Inclination for second harmonic perturbations is:

$$\delta i_k = C_{ic} \cdot \cos(2\Phi_k) + C_{is} \cdot \sin(2\Phi_k) \tag{B.13}$$

The corrected Argument of Latitude is:

$$u_k = \Phi_k + \delta u_k \tag{B.14}$$

The corrected Radius is:

$$r_k = A \cdot (1 - e \cdot \cos(E_k)) + \delta r_k \tag{B.15}$$

The corrected Inclination is:

$$i_k = i_0 + \delta i_k + IDOT \cdot t_k \tag{B.16}$$

The position in Orbital Plane is:

$$\begin{aligned} x_k &= r_k \cdot \cos(u_k) \\ y_k &= r_k \cdot \sin(u_k) \end{aligned} \tag{B.17}$$

The corrected Longitude of Ascending Node is:

$$\Omega_k = \Omega_0 + (\dot{\Omega} - \dot{\Omega}_e) \cdot t_k - \dot{\Omega}_e \cdot t_{oe} \tag{B.18}$$

The position of the satellite Earth Centered Earth Fixed (ECEF) coordinates is:

$$\begin{aligned} x_k &= x_k \cdot \cos(\Omega_k) - y_k \cdot \cos(i_k)\sin(\Omega_k) \\ y_k &= x_k \cdot \sin(\Omega_k) + y_k \cdot \cos(i_k)\cos(\Omega_k) \\ z_k &= y_k \cdot \sin(i_k) \end{aligned} \tag{B.19}$$

The calculated satellite coordinates that are obtained from the above equations have to be corrected for the earth rotation effect. This correction is taken into account because during the time of signal propagation, the earth rotates. The angle of this rotation can be calculated using the difference between the time of transmission and the estimated time of reception and the average earth's rotation rate $\dot{\Omega}$.

This rotation angle can be given in an equation form as:

$$\gamma = (\text{Est\_TOR} - \text{TOT}) * \dot{\Omega} \tag{B.20}$$

with:

$\dot{\Omega}$	the average earth rotation	(radians/seconds)
Est_TOR	the estimated time of reception	(seconds)
TOT	the time of transmission	(seconds)

This rotation angle can be used to calculate the new position of the satellite which is corrected for the earth rotation. This is a rotation arround the positive (ECEF) z-axis. This correction is given in litt [3] as:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}_{\text{Est\_TOR}} = \begin{bmatrix} \cos\gamma & \sin\gamma & 0 \\ -\sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{\text{TOT}} \tag{B.21}$$

Appendix C The MX4200D Multi-port Interface

The MX4200D multi-port interface consists of four serial, bi-directional data ports. These ports with the accompanying general function are:

- Control Port : as controller
- Equipment Port : as autopilot, radar, satellite communications terminal and Loran-C receiver
- Raw Data Port : for data logging
- RTCM Port : for differential corrections

The ports used here are the control port and the raw data port. These ports will be described in the following sections. The equipment port is used to allow the MX4200D to communicate with external equipment which have an interface using the NMEA-0183 electrical standard for serial data. Standard NMEA-0183 sentences containing position, track and speed are the periodically output of this port. The RTCM port is used by the MX4200D. This port is typically connected to a modem data link for the reception of the differential corrections from a remote GPS differential reference station.

C.1 Control Port

The MX4200D can be programmed with some messages of the control port. The messages, their labels and the structure which are used here are listed in the Table C.1.

The functions of the given sentences are:

$PMVXG,000 Part A of the Initialization/Mode Control. It initializes the time, position and antenna height of the MX4200

$PMVXG,001 Part B of the Initialization/Mode Control. It specifies various navigation parameters: Altitude aiding, acceleration, maximum DOP limits, and satellite elevation limits.

$PMVXG,018 Restart Control. It causes the MX4200 to terminate operation and restart in either the warm or cold state, as specified.

$PMVXG,024 Raw Data Port Record Selection. This messages controls which data records are output on the Raw Data port.

$PMVXG,027 Raw Data Port Control Settings. This sentence is used to define the serial configuration of the Raw Data port. In conjunction with PMVXG,024, it defines in

which format raw measurement data is output to the Raw Data port, and whether to output almanac and ephemeris data.

Table C.1 Magnavox MX4200D Control Port Input Sentence

Rec.number	Field	Field Contents	Range
$PMVXG,000	DD	Day of month	1-31
	MM	Month of year	1-12
	YYYY	Year	1991-9999
	HHMMSS	GMT	235959
	DDMM.MMMM	Latitude (degrees, minutes)	0-89.9999
	N	North (S = South)	N or S
	DDDMM.MMMM	Longitude (degrees, minutes)	0-179.999
	W	West (E = East)	E or W
	HHHHH.H	Altitude (meters)	0-99999.9
	X	Not used	
$PMVXG,001	X	Constrain Altitude: 0 = never 1 = auto 2 = always 3 = coast	0-3
	X	Not used	
	X.XX	Horizontal acceleration factor (m/s^2)	0.50-10.0
	X	Not used	
	XXXX	VDOP limit	1-9999
	XXXX	HDOP limit	1-9999
	XX	Elevation limit	0-90
	X	Time output mode: U = UTC, L = Local time	U, L
	HHHMM	Local time offset (±) from GMT	± 0-2359
$PMVXG,018	X	Reset type: W = Warm start; C = Cold start; T = Tepid start	C, W, T
$PMVXG,024	X	Nav Result: + = enable output, - = disable output	+, -
	X	Raw Measurements	+, -
	X	Almanac and Ephemeris	+, -
	X	Constellation Information	+, -
	X	Time Recovery	+, -
	X	Full Debug	+, -
	X	Partial Debug	+, -
$PMVXG,027	XX	Input Baud Rate Selection	
	X	Parity/No. Data Bits Selection	
	XX	Raw Measurement Record Compression Control: 0 = ASCII; 1 = Compressed only; 2 = Both	0-2
	X	Almanac/Ephemeris Request: 0 = Output almanac now 1 = Output ephemeris data now 2 = Output almanac and ephemeris data now	0-2

C.2 Raw Data Port

The raw data records consist of ASCII characters terminated by a carriage return/line feed. The raw data lines with record type 1 and 2 contain the raw satellite data. The contents of these data are given in chapter 3.2.

Table C.2 Raw Data Port Records

Rec.type	Byte	Identifier	Description	Range
100	1-4	ID	Record ID	100
	6-7	PRN	Satellite PRN number used for Almanac	1-32
101-132	1-4	ID	Record ID	101-132
	6-76		Almanac data for satellites 1-32	
133-134	1-4	ID	Record ID	133-134
	6-76		Health indicators for satellites 1-32	
135	1-4	ID	Record ID	135
	6-76		Ionospheric correction	
136-150			Reserved for special messages, spares and additional data	
200	1-4	ID	Record ID	200
	6-7	PRN	Satellite PRN number of the messages in rec.type 201-203	1-32
201	1-4	ID	Record ID	201
	6-76	Subframe 1	The contents is according to ICD-GPS-200. It contains 24 pairs of hex-ASCII digits.	
202	1-4	ID	Record ID	202
	6-76	Subframe 2	id for subframe 2	
203	1-4	ID	Record ID	203
	6-76	Subframe 3	id for subframe 3	
411-416	1-4		Record Type: 411 = channel 1 412 = channel 2 413 = channel 3 414 = channel 4 415 = channel 5 416 = channel 6	411-416
	6-7		Satellite PRN	1-32
	9-11		Azimuth to satellite, in degrees	0-359
	13-14		Elevation of satellite, in degrees	0-90
	16-22		GPS time in seconds	0-604800

Appendix D Software listing

```
(* ----------------------- FILE MAGNAVOX.PAS --------------------------- *)

PROGRAM magnavox;

(* ************************************************************************ *)
(*                                                                          *)
(*   This is the main program of the User Interface                         *)
(*           for the Magnavox MX4200D Receiver                              *)
(*                                                                          *)
(* ************************************************************************ *)

USES
   crt,
   types,
   input,
   convdata,
   rec_pos;


PROCEDURE show_setting_choises (VAR state : CHAR);
VAR
   key : CHAR;

BEGIN
   CLRSCR;
   writeln;
   writeln;
   writeln(' Program for MX 4200D GPS Receiver ');
   writeln;
   writeln(' 1 : Convert the raw data to files');
   writeln;
   writeln(' 2 : User position calculation');
   writeln;
   writeln(' 3 : exit');
   writeln;

   key := ReadKey;
   state := key
END;


PROCEDURE test (VAR state : CHAR);
BEGIN
   CASE state OF
    '1': BEGIN
           convert_data_from_receiver (f_in, AllChannelMeas, Navdata);
           gotoxy (5,23);
           write('press RETURN to go back to menu');
           readln;
         END;

    '2': BEGIN
           begin_user_position;
           receiver_position (f_in, AllChannelMeas, result, RECpos);
           gotoxy (5,23);
           write('press RETURN to go back to menu');
           readln;
         END;

    '3': readln;

   END;
END;


BEGIN
   show_setting_choises (state);
   WHILE state <> '3'  DO
   BEGIN
     test (state);
     show_setting_choises (state);
   END;
END.

(* ------------------------ END OF FILE ------------------------------- *)
```

```
(* ---------------------- FILE CONVDATA.PAS --------------------------- *)

UNIT convdata;

(* ************************************************************************ *)
(*                                                                          *)
(*   Function : interface part of the user position calculation item        *)
(*                                                                          *)
(* ************************************************************************ *)

INTERFACE
{$N+}

USES
   crt,
   types;

PROCEDURE convert_data_from_receiver (VAR f_in           : TEXT;
                                      VAR AllChannelMeas : AllChannelMeasType;
                                      VAR NavData        : NavDataType);

IMPLEMENTATION

USES
   common,
   files,
   input,
   MX4200;


PROCEDURE convert_data_from_receiver (VAR f_in           : TEXT;
                                      VAR AllChannelMeas : AllChannelMeasType;
                                      VAR NavData        : NavDataType);
(* ------------------------------------------------------------------------ *)
(*                                                                          *)
(*   INPUT : f_in : input file which come from the receiver                 *)
(*   OUTPUT : AllChannelMeas : the record of the converted raw sat. data    *)
(*                    NavData : the ephemeris data                          *)
(*                       futt : the output file                             *)
(*                                                                          *)
(* ------------------------------------------------------------------------ *)

VAR
   line        : STRING;
   user_ms_int : longint;

BEGIN    { of PROCEDURE convert_data_magnavox_receiver }
   output_of_magnavox (input_file_name);          {screen information to user}
   open_files_for_magnavox (input_file_name, output_file_name);
                                                          {open all of files}
   BuildUpScreen;

   WHILE (NOT eof(f_in) AND (last_meas_no <> end_meas_no)) DO
   BEGIN
      readln (f_in, line);                                {read the input file}

      unit_4200 (line);                                   {process the input file}
                                          {write the converted raw data to files}
      IF (RecNum = '   2') THEN
        write_rawdata_to_file (AllChannelMeas, futt);
      IF (RecNum = ' 203') THEN
        write_navdata_to_file (SV20x, last_meas_no, NavDataFiles, NavData);

      user_ms_int := round(AllChannelMeas.user_ms/1000);
                {print the measurement number and the raw data to the screen}
      PrintScreenCurrentMeasNumber (user_ms_int);
      PrintScreenMeasData (AllChannelMeas);

      IF (last_meas_no = start_meas_no) THEN reopen_output_files;
   END;
   close_files_for_magnavox;                          {close all of the files}
END;     { of PROCEDURE convert_data_magnavox_receiver }

END.

(* ------------------------ END OF FILE ------------------------------- *)
```

```
(* ----------------------- FILE REC_POS.PAS --------------------------- *)

UNIT rec_pos ;

(* ************************************************************************ *)
(*                                                                          *)
(*   Function : interface part of the user position calculation             *)
(*                                                                          *)
(* ************************************************************************ *)

INTERFACE
{$N+,E+}

USES
   types,
   input,
   files,
   common,
   mx4200,
   calc_sat,
   calc_pos;


PROCEDURE receiver_position (VAR f_in           : TEXT;
                             VAR AllChannelMeas : AllChannelMeasType;
                             VAR Result         : ResultType;
                             VAR RECpos         : ResultType);

IMPLEMENTATION

PROCEDURE receiver_position (VAR f_in           : TEXT;
                             VAR AllChannelMeas : AllChannelMeasType;
                             VAR result         : ResultType;
                             VAR RECpos         : ResultType);
(* ------------------------------------------------------------------------ *)
(*                                                                          *)
(*   Function :                                                             *)
(*   Input : f_in : input file contains the raw satellite data              *)
(*   Output :                                                               *)
(*      AllChannelMeas : the uncompressed form of the raw satellite data    *)
(*               Result : the corrected transmission time                   *)
(*               RECpos : the user coordinates in ECEF                      *)
(*                                                                          *)
(* ------------------------------------------------------------------------ *)
VAR
   line : STRING;
   user_ms_int : longint;
   i : byte;

BEGIN
   reset_state_condition;                       {reset the condition of state}
   input_rec_position
            (input_file_name, output_file_name, channels_in_input, state);

   BuildUpScreen;
   buildup_screen_position;                                {build up of the screen}

   open_files_rec_position (input_file_name, output_file_name);
                                                   {open input and output files}
{ -------------------- begin of processing data --------------------------- }
   WHILE (NOT eof(f_in) AND (last_meas_no <> end_meas_no)) DO
   BEGIN
      readln (f_in, line);
   {--- process a line of data ---}
      unit_4200 (line);
   {--- write to file ---}
      IF (RecNum = '   2') THEN
           write_rawdata_to_file (AllChannelMeas, futt);
      IF (RecNum = ' 203') THEN
      BEGIN
        navdata_to_arraynavdata (SV20X, Navdata, ArrayNavData);
        write_navdata_to_file (SV20X, last_meas_no, NavDataFiles, NavData);
      END;

      AllChannelMeas.user_ms := trunc(AllChannelMeas.user_ms/1000);

  {--- corrected the transmissie time ---}
      CorrectTOT (channels_in_input, AllChannelMeas, ArrayNavData, result);

  {--- begin of calculation position ---}
      IF (RecNum = '   2') THEN
      BEGIN
         calc_position (num_of_sv, channels_in_input, result, RECpos, Q, SVpos);
         write_to_file_rec (num_of_sv, RECpos);
         PrintScreenCurrentMeasNumber (result.UncTOR);
         PrintScreenMeasData (AllChannelMeas);
         print_screen_user_position(RECpos);
      END;
      IF (last_meas_no = start_meas_no) THEN reopen_output_files;

   END;       { of WHILE (NOT eof(f_in) AND (last_meas_no <> end_meas_no)) DO}

   close_files_rec_position;
END;

END.

(* ------------------------ END OF FILE ------------------------------- *)
```

```
(* ------------------------ FILE MX4200.PAS ---------------------------- *)

UNIT MX4200;

(* ************************************************************************ *)
(*                                                                          *)
(*   Function : Receive the MX4200 datastring                               *)
(*              Putting the converted data to the record                    *)
(*                                                                          *)
(* ************************************************************************ *)

INTERFACE
{$N+,E+}

USES
   crt,
   common,
   types;

PROCEDURE unit_4200 (VAR Rec : STRING);

(* ------------------------------------------------------------------------ *)
(*                                                                          *)
(*  Function : convert the compressed form of data                          *)
(*                                                                          *)
(*  input  : Rec: row of data from the MX4200                               *)
(*                                                                          *)
(*  output : AllChannelMeas : the converted raw data from rec type 2        *)
(*                   Navdata : the ephemeris data                           *)
(*                                                                          *)
(* ------------------------------------------------------------------------ *)

IMPLEMENTATION

CONST
   LengthOf2   = 77;                { Check for the minimum Length of record }
   LengthOf200 = 8;                     { messages from mx4200 without CR/LF }
   LengthOf201 = 76;
   LengthOf202 = 76;
   LengthOf203 = 76;

TYPE
   Two2PowerType  = ARRAY [-60..-1] OF float;

VAR
   Two2Power        : Two2PowerType;
   Last20x          : byte;          { last line 200..202, 0 zero if nothing }
   loop             : integer;
   StoredHatchData  : StoredHatchDataType;


PROCEDURE CalcOnesTwo2Power ( VAR Two2Power : Two2PowerType );
(* ------------------------------------------------------------------------ *)
(*   Calculates once all powers of two, that used in MX4200.                *)
(* ------------------------------------------------------------------------ *)
VAR i : integer;

BEGIN
   Two2Power[-1]:=0.5;
   FOR i := -2 DOWNTO -60 DO Two2Power[i] := Two2Power[i+1]/2;
END;


PROCEDURE ConvertRec20xStringToNewString ( VAR Rec : STRING );
(* ------------------------------------------------------------------------ *)
(*   This procedure converts the MX4200 line of e.g. the records 201,       *)
(*   202 and 203 containing hexadecimal information grouped in bytes        *)
(*   (2 characters) seperated by a space in one line of only characters.    *)
(*   So FROM:                                                               *)
(*     (5 spaces,2 hex chars,1 space, 2 hex chars, 1 space etc.) (24x)      *)
(*   TO:                                                                    *)
(*     (2 characters*24bytes)=48                                            *)
(*   Each GPS word contains 30 bits: 24 bits of infon and 6 bits parity     *)
(*   From every message contains the words 3 to 10, so 8 words.             *)
(*   Two hex characters is one byte is 8 bits. One word is 6 characters.    *)
(*   So one message is 8*6=48 characters.                                   *)
(*   Pre : string of 76 characters                                          *)
(*   Post: string of 48 characters                                          *)
(*         Input : Rec                                                      *)
(*         Output: Rec; string of 48 characters (=24 bytes)                 *)
(*         Uses  : -                                                        *)
(* ------------------------------------------------------------------------ *)
VAR
   NewRec  : STRING;                                      { local substring }
   x       : byte;                                               { counter }

BEGIN
   NewRec:='';                                                { Reset NewRec }
   FOR x := 1 TO 24 DO
      NewRec:=NewRec + copy(Rec, 3+3*x, 2); { Add new byte, start at place 6 }
   Rec:=NewRec;
END;          { of PROCEDURE ConvertRec20xStringToNewString }


FUNCTION ValueOfBits (Rec           : STRING;
                      FirstBit,
                      NumberOfBits  : byte;
                      TwosComplement : boolean) : longint;
(* ------------------------------------------------------------------------ *)
(*  This function returns the value of the bits starting place with Firstbit*)
(*  and with length NumberOfBits. If TwosComplement is true                 *)
(*  the bits are seen in twoscomplement notation, otherwise not.            *)
(*  SO: INPUT  : string of a long hexadecimal characters without spaces     *)
(*      OUTPUT : value of of part of this string of characters              *)
(*                                                                          *)
(*  In combination with ConvertRec20xStringToNewString this gives:          *)
(*  The firstbit of word3 in navigation message is FirstBit=1               *)
(*  For every next word 24 must be added to FirstBit                        *)
(*  So the lastbit is 8*24=192                                              *)
(*                                                                          *)
```

```
111 (*  RESTRICTIONS: The maximum number of bits (NumberOfBits) is:            *)
112 (*   32 if twoscomplement and contains 8 characters only Firstbit/4=integer *)
113 (*   28 if twoscomplement and contains 9 characters (Firstbit/4<>integer)   *)
114 (*   30 if not twoscomplenent                                               *)
115 (*         Input : Rec: string of hexadecimal characters                    *)
116 (*                 Firstbit  (MSB)                                          *)
117 (*                 NumberOfBits                                             *)
118 (*                 TwosComplement : boolean                                 *)
119 (*         Output: value of bits (longint)                                  *)
120 (*         Uses  : -                                                        *)
121 (* ------------------------------------------------------------------------ *)
122 VAR
123    Value,
124    TmpValue          : longint;                  { used to store $FFFFFFFF }
125    LastBit,                                       { contains number of LSB }
126    FirstChar,                                  { number of char containing MSB }
127    LastChar,                                   { number of char containing LSB }
128    NumberOfChar,                                  { number of characters used }
129    ShiftRight,                                 { used to right align the bits }
130    TmpByte           : byte;                        { used for sign-checking }
131    ValError          : integer;{error returning from val-function, not used }
132
133 BEGIN
134    LastBit      := FirstBit+NumberOfBits-1;          { range of bits [1..192] }
135    FirstChar    := trunc ((FirstBit-1)/4) + 1;     { characters range [1..48] }
136    LastChar     := trunc (( LastBit-1)/4) + 1;
137    NumberOfChar := LastChar-FirstChar+1;
138    ShiftRight   := LastChar*4-LastBit;
139                          { how many shifts to the right for right alignment }
140    val('$'+copy(Rec,FirstChar,NumberOfChar),Value,ValError);
141                                                 { Value := substring of Rec }
142    Value        := Value SHR ShiftRight;
143    TmpValue     := $FFFFFFFF;                { used in cover/shift commandos }
144                       { split twoscomplement, positive and negative numbers }
145    IF TwosComplement
146    THEN BEGIN                                     { twoscomplement notation }
147       val ( '$'+copy(Rec,FirstChar,1) , TmpByte , ValError  );
148                                                      { TmpByte := FirstChar }
149       TmpByte:= TmpByte SHR (FirstChar*4-FirstBit);
150                                               { shift Firstbit to LSB postion }
151       TmpByte:= TmpByte AND $01;                      { mask all other bits }
152
153       IF (TmpByte=0)                                      { check Firstbit }
154
155       THEN BEGIN                                { sign-bit=0; positive number }
156          Value:= Value AND (TmpValue SHR (32-NumberOfBits));{cover left bits }
157       END      {000011111}                                  { not encountered }
158
159       ELSE BEGIN                                { sign-bit=1; negative number }
160          TmpValue := TmpValue SHR (32-NumberOfBits);
161          TmpValue := NOT TmpValue;
162          Value := Value OR TmpValue;                 { set left bits to 1 }
163       END;     {1111100000}
164    END
165    ELSE BEGIN  {not twoscomplement}
166       Value:= Value AND (TmpValue SHR (32-NumberOfBits));  { cover left bits }
167    END;        {00000111111}                                { not encountered }
168                                          { End splitting for different numbers }
169    ValueOfBits:=Value;
170 END;        { of FUNCTION ValueOfBits }
171
172
173 PROCEDURE UnMap78 ( VAR Buf        : STRING;
174                     CheckSum_Index : integer;
175                     VAR Succes     : boolean );
176 (* ------------------------------------------------------------------------ *)
177 (*  See Magnavox guide for explanation of this procedure.                   *)
178 (*  Index over all of ;the packed adata bytes and the stored checksum,      *)
179 (*  map them back to binary. Test for special case of rubout/delete.        *)
180 (*  compute checksum.)                                                      *)
181 (*  INPUT : Buf                                                             *)
182 (*          CheckSum_Index                                                  *)
183 (*  OUTPUT :Buf                                                             *)
184 (*          Succes                                                          *)
185 (*  USED BY: get2 only                                                      *)
186 (* ------------------------------------------------------------------------ *)
187 VAR
188    i        : integer;
189    CheckSum : word;
190
191 BEGIN
192    CheckSum := 0;
193    FOR i := 1 TO CheckSum_Index DO
194    BEGIN
195       IF (Buf[i] = chr($f7))
196          THEN Buf[i]:= chr($7f);
197       buf[i]:= chr( ord(Buf[i]) - 37);
198       CheckSum:= CheckSum XOR ord( Buf[i]); {!!!!!}
199    END;
200    Succes := (CheckSum=0);
201 END;
202
203
204 PROCEDURE UnPack78(     Buf            : STRING;
205                         Num_Bits       : integer;
206                     VAR Out_Value      : longint;
207                     VAR Total_Num_Bits : integer);
208 (* ------------------------------------------------------------------------ *)
209 (*  See Magnavox guide for explanation of this procedure.                   *)
210 (*  INPUT : Buf                                                             *)
211 (*          Num_Bits                                                        *)
212 (*          Total_Num_Bits                                                  *)
213 (*  OUTPUT: Out_Value                                                       *)
214 (*          Total_Num_Bits                                                  *)
215 (*  USED BY: procedure Get2 only                                            *)
216 (* ------------------------------------------------------------------------ *)
217 VAR
218    bit_index,
219    buf_index,
220    num_bits_remaining : integer;
```

```
221    tmp_value          : longint;
222
223 BEGIN
224    IF (total_num_bits = 0)       { start on new buffer if caller requests it }
225    THEN BEGIN
226       buf_index:= 1;
227       bit_index:= 6;
228    END;
229    num_bits_remaining:= num_bits;
230    out_value:= 0;
231
232    WHILE (num_bits_remaining > 0) DO
233    BEGIN
234       num_bits_remaining := num_bits_remaining -1;
235    { unpack the next bit and 'or' it into the proper bit in the output value }
236       tmp_value := ord( buf[ buf_index]);
237       tmp_value := tmp_value SHR bit_index;
238       tmp_value := tmp_value AND 1;
239       tmp_value := tmp_value SHL num_bits_remaining;
240       out_value := out_value OR tmp_value;
241       bit_index := bit_index - 1;
242       IF (bit_index < 0)
243       THEN BEGIN
244          bit_index := 6;
245          buf_index := buf_index + 1;
246       END;
247    END;
248    total_num_bits := total_num_bits + num_bits;
249 END;
250
251
252 PROCEDURE Get2 (    Rec : STRING;
253                  VAR AC  : AllChannelMeasType );
254 (* ------------------------------------------------------------------------ *)
255 (*  This procedure reads the compressed record type 2 and puts its          *)
256 (*  information in AC.                                                      *)
257 (*  It always resets AC.                                                    *)
258 (*        Input  : Rec                                                      *)
259 (*        Output : AC                                                       *)
260 (*        USES   : UnMap and UnPack                                         *)
261 (* ------------------------------------------------------------------------ *)
262 VAR
263    Succes          : boolean;                  { indicates if UnMap succeeded }
264    TotalBits       : integer;
265    i               : 1..max_channels;
266    DataValid,
267    OutOfLock       : boolean;
268    prn_number      : 0..max_sv_id;
269    raw_code,
270    cas_code        : integer;
271    cn0             : byte;
272    long_temp,
273    sub_ms_phase,
274    chnl_time_ms,
275    ch_ms_base,
276    user_time_ms    : longint;
277    phase_fraction,
278    costas_ratio    : shortint;
279    inverted_data   : integer;
280    temp_ms,
281    temp_phase      : float;
282    MeasMisLoc      : longint;
283
284 BEGIN        { of PROCEDURE Get2 }
285    ResetAllChannelMeasType(AC);
286
287    IF (length(Rec)>=LengthOf2)             { check minimum length of message }
288    THEN BEGIN
289       Rec := copy( Rec, 6, length(Rec) - 5 );    { remove first 5 characters }
290       UnMap78( Rec, length(Rec), Succes);
291       IF Succes
292       THEN BEGIN
293
294       {**** c o m m o n   f o r   a l l   m e a s u r e m e n t s *****}
295          TotalBits := 0;          { unpack items recorded once for each set }
296          UnPack78( Rec, 30, user_time_ms, TotalBits);
297          UnPack78( Rec, 30, ch_ms_base, TotalBits);
298          Unpack78( Rec, 22, long_temp, TotalBits);
299          AC.user_ms := user_time_ms;                 { store common usertime }
300          last_meas_no := trunc(AC.user_ms/1000);
301
302       { * * * *  f o r  e a c h  c h a n n e l  * * * * }
303          FOR i := 1 TO max_channels DO
304          BEGIN
305             UnPack78( Rec, 1, long_temp, TotalBits);
306              DataValid:= (long_temp=1);
307             UnPack78( Rec, 6, long_temp, TotalBits);
308              prn_number:= long_temp;
309             UnPack78( Rec, 5, long_temp, TotalBits);
310              chnl_time_ms:= ch_ms_base + long_temp;
311             UnPack78( Rec, 21, long_temp, TotalBits);
312              sub_ms_phase:= long_temp;
313             UnPack78( Rec, 8, long_temp, TotalBits);
314              phase_fraction:= shortint(long_temp - 128);
315             UnPack78( Rec, 10, long_temp, TotalBits);
316              raw_code:= integer( long_temp - 512);
317             UnPack78( Rec, 10, long_temp, TotalBits);
318              cas_code:= integer( long_temp - 512);
319             UnPack78( Rec, 4, long_temp, TotalBits);
320              costas_ratio:= shortint(long_temp * 16 - 128);
321             UnPack78( Rec, 5, long_temp, TotalBits);
322              cn0 := byte( long_temp + 25);
323             UnPack78( Rec, 2, long_temp, TotalBits);
324              inverted_data:= integer(long_temp);
325             IF DataValid
326             THEN WITH AC.channel[i] DO BEGIN
327                PRN        := prn_number;
328                phi        := sub_ms_phase;
329                phi_frac   := phase_fraction;
330                temp_ms    := ((user_time_ms - chnl_time_ms)/1000) * light_speed;
```

```
331              temp_phase := -((sub_ms_phase+(phase_fraction/256)) * L1_TO_VEL);
332              int_phase  := temp_ms + temp_phase;
333              RawCODE    := raw_code;
334              pr_receiv  := int_phase-RawCODE/L1_TO_VEL;
335              SNR        := cn0;
336              IF (inverted_data = 0)
337              THEN Ambiguity:='?'
338              ELSE IF (inverted_data = 1)
339                   THEN Ambiguity:='+'
340                   ELSE IF (inverted_data = 2)
341                        THEN Ambiguity:= '-';
342              OutOfLock:=(Ambiguity='?');
343              IF (NOT OutOfLock)
344                THEN OutOfLock := ( Ambiguity <>
345                                    StoredHatchData[prn_number].LastAmbiguity );
346              IF OutOfLock
347                THEN BEGIN
348                     NoOfHatch := 1;
349                     HatchCODE := RawCODE;
350                     END
351                ELSE BEGIN       {not Out Of Lock}
352                     IF (StoredHatchData[prn_number].NumberOfHatch>=HatchFil)
353                     THEN NoOfHatch := HatchFil
354                     ELSE NoOfHatch :=
355                              StoredHatchData[prn_number].NumberOfHatch+1;
356                     HatchCODE := StoredHatchData[prn_number].LastHatchCode
357                                  *(NoOfHatch-1)/NoOfHatch+RawCODE/NoOfHatch;
358                END;
359                PseudoRange         := int_phase - HatchCODE * L1_TO_VEL;
360                time_of_transmission :=
361                              (user_time_ms/1000) - PseudoRange/light_speed;
362
363     {* * * * *  Houdt Hatchfilter parameters bij * * * * *}
364                StoredHatchData[prn_number].user_ms       := AC.user_ms;
365                StoredHatchData[prn_number].NumberOfHatch := NoOfHatch;
366                StoredHatchData[prn_number].LastHatchCODE := HatchCODE;
367                StoredHatchData[prn_number].LastAmbiguity := Ambiguity;
368             END;  {DataValid and with AC.   do}
369          END;     {for channel}
370     END;  {of if succes UnMap}
371   END;  {enough length}
372 END;    { of PROCEDURE Get2 }
373
374
375 PROCEDURE Get8 ( Rec : STRING );
376 BEGIN
377    writeln ( fpos, Rec );
378 END;    { of PROCEDURE Get8 }
379
380
381 PROCEDURE Get200 (    Rec : STRING;
382                   VAR ND  : NavDataType );
383 (* ------------------------------------------------------------------------ *)
384 (*  A 200 line is in Rec. It fills ND: reads the satellite number
385     from the 200 line and puts it in SV20x:=PRN, Last20x:=200;
386     ERROR: If the line is not correct or the satellite number can't be
387            detected then Last20x:=0;
388     Input : Rec
389     Output: ND                                                          *)
390 (* ------------------------------------------------------------------------ *)
391 VAR
392    fault : integer;
393    Satno : byte;
394
395 BEGIN
396    IF (length(Rec) >= LengthOf200)        { check minimum length of message }
397    THEN BEGIN                                    { 200 line is long enough }
398       val ( copy( Rec, 6, 3), Satno, fault);              { Get SV number }
399       IF ( (fault=0) AND ((Satno>=1) AND (Satno<=max_sv_id)) )
400       THEN BEGIN                          { satellite number correct detected }
401          SV20x   := Satno;
402          Last20x := 200;
403       END
404       ELSE BEGIN
405           gotoxy(5,19);
406          writeln('Wrong satellite number detected, rec. no. 200');
407          Last20x := 0;
408       END;
409    END
410    ELSE BEGIN
411       gotoxy(5,20);
412       writeln('Rec. no 200 not long enough');
413       Last20x := 0;
414    END;
415 END;         { of PROCEDURE Get200 }
416
417
418 PROCEDURE Get201 (    Rec : STRING ;
419                   VAR ND  : NavDataType);
420 (* ------------------------------------------------------------------------ *)
421 (*  PRE   : Last20x=200
422     POST  : Last20x=201 or 0 if error
423     Input : Rec
424     Output: ND
425     Uses  : ValueOfBits
426             ConvertRec20xStringToNewString                              *)
427 (* ------------------------------------------------------------------------ *)
428 BEGIN
429    IF ( (length(Rec)>=LengthOf201) AND (Last20x=200) )
430    THEN BEGIN                          { PREconditions and Rec is long enough }
431       ConvertRec20xStringToNewString(Rec);  { Rec is string of 48 characters }
432       WITH ND DO BEGIN
433          WeekNo   := ValueOfBits(Rec,       1, 10, False);
434          Accuracy := ValueOfBits(Rec,      13,  4, False);
435          Health   := ValueOfBits(Rec,      17,  6, False);
436          IODC_MSB := ValueOfBits(Rec,      23,  2, False);
437          Tgd      := ValueOfBits(Rec, 4*24+17,  8,  True)*Two2Power[-31];{sec}
438          IODC     := ValueOfBits(Rec,  5*24+1,  8, False);  {Last 8 bits only}
439          toc      := ValueOfBits(Rec,  5*24+9, 16, False)*16;           {sec}
440          af2      := ValueOfBits(Rec,  6*24+1, 8 ,  True)*Two2Power[-55];{s/s^2}
```

```
441          af1      := ValueOfBits(Rec,  6*24+9, 16,  True)*Two2Power[-43];{s/s}
442          af0      := ValueOfBits(Rec,  7*24+1, 22,  True)*Two2Power[-31];{sec}
443                {CHECKED with STANAG: OK}
444          END;                                        { end with AR.TmpNavData^ do}
445       Last20x:=201;
446       END
447    ELSE BEGIN                       { not PREconditions or Rec not long enough }
448       Last20x:=0;
449       gotoxy(5,21);
450       writeln('Error in rec. no. 201');
451       END;
452 END;       { of PROCEDURE Get201 }
453
454
455 PROCEDURE Get202 (    Rec : STRING;
456                   VAR ND  : NavDataType );
457 (* ------------------------------------------------------------------------ *)
458 (*  PRE : Last20x=201
459     POST: Last20x=202 or 0
460     Input : Rec, ND
461     Output: ND
462     Uses: ValueOfBits
463           ConvertRec20xStringToNewString
464     NOTE: according to the ICD-GPS-200 specifications the value of pi
465     used to convert semicircles to radians = 3.1 415 926 535 898           *)
466 (* ------------------------------------------------------------------------ *)
467 BEGIN
468    IF ( (length(Rec)>=LengthOf202) AND (Last20x=201) )
469    THEN BEGIN                             { PREconditions and Rec is long enough }
470       ConvertRec20xStringToNewString(Rec);  { Rec is string of 48 characters }
471       WITH ND DO BEGIN                                    { Afrondfouten!!!!! }
472          IODE2  := ValueOfBits(Rec,       1,  8, False);
473          Crs    := ValueOfBits(Rec,       9, 16,  True)*Two2Power[-5];     {m}
474          deltan := ValueOfBits(Rec,  1*24+1, 16,  True)*Two2Power[-43]*pi_SEMI;
475          Mo     := ValueOfBits(Rec, 1*24+17, 32,  True)*Two2Power[-31]*pi_SEMI;
476          Cuc    := ValueOfBits(Rec,  3*24+1, 16,  True)*Two2Power[-29];  {rad}
477          e      := ValueOfBits(Rec, 3*24+17, 16,  False)*Two2Power[-17];
478          e := e + ValueOfBits(Rec, 4*24+9, 16, False)*Two2Power[-33];{dim. less}
479                  { If e > 0.03 }                                 {Check on range}
480          Cus    := ValueOfBits(Rec,  5*24+1, 16,  True)*Two2Power[-29];  {rad}
481          Asqrt  := ValueOfBits(Rec, 5*24+17, 16, False)*Two2Power[-3];
482          Asqrt  := Asqrt + ValueOfBits(Rec, 6*24+9, 16, False)*Two2Power[-19];
483        { For calculation of Asqrt and e the maximum value of Getvalue of bits
484                                  is reached. Therefore it is split in two parts}
485          toe    := ValueOfBits(Rec,  7*24+1, 16, False)*16;              {sec}
486                    {CHECKED WITH STANAG: OK}
487          END;                                        { end with AR.TmpNavData^ do}
488       Last20x:=202;
489       END
490    ELSE BEGIN                          {not PREconditions or Rec not long enough}
491       Last20x:=0;
492       gotoxy(5,22);
493       writeln('Error in converting 202')
494    END;
495 END;           { end PROCEDURE Get202 }
496
497
498 PROCEDURE Get203 (    Rec : STRING;
499                   VAR ND  : NavDataType );
500 (* ------------------------------------------------------------------------ *)
501 (*  PRE : AR.Last20x=202
502     POST: AR.Last20x=0
503     Input : Rec
504             ND
505     Output: ND
506     Uses  : ValueOfBits
507             ConvertRec20xStringToNewString
508     NOTE: according to the ICD-GPS-200 specifications the value of pi
509           used to convert semicircles to radians = 3.1 415 926 535 898      *)
510 (* ------------------------------------------------------------------------ *)
511 BEGIN
512    IF ( (length(Rec)>=LengthOf203) AND (Last20x=202) )
513    THEN BEGIN                              {PREconditions and Rec is long enough}
514       ConvertRec20xStringToNewString(Rec);    {Rec is string of 48 characters}
515       WITH ND DO BEGIN                                        {Afrondfouten!!!!}
516          Cic    := ValueOfBits(Rec,       1, 16, True)*Two2Power[-29];    {rad}
517          omegao:= ValueOfBits(Rec,      17, 32, True)*Two2Power[-31]*pi_SEMI;
518          Cis    := ValueOfBits(Rec,  2*24+1, 16, True)*Two2Power[-29];    {rad}
519          io     := ValueOfBits(Rec, 2*24+17, 32, True)*Two2Power[-31]*pi_SEMI;
520          Crc    := ValueOfBits(Rec,  4*24+1, 16, True)*Two2Power[-5];       {m}
521          omega  := ValueOfBits(Rec, 4*24+17, 32, True)*Two2Power[-31]*pi_SEMI;
522          omegadot:= ValueOfBits(Rec,6*24+1, 24, True)*Two2Power[-43]*pi_SEMI;
523          IODE3  := ValueOfBits(Rec,  7*24+1,  8,False);
524          IDOT   := ValueOfBits(Rec,  7*24+9, 14, True)*Two2Power[-43]*pi_SEMI;
525                       {CHECKED WITH STANAG: OK}
526       END;                                          { end with AR.TmpNavData^ do }
527       Last20x:=0;
528       {IF (ND.IODC=ND.IODE2) AND (ND.IODC=ND.IODE3)
529       THEN PrintScreenNewNavData(last_meas_no, SV20x, ND.IODC)
530       ELSE writeln ('Error Navigation data');}
531
532    END
533    ELSE BEGIN      {not PREconditions or Rec not long enough}
534       Last20x:=0;
535       gotoxy(5,23);
536       writeln('Error in 203');
537    END;
538 END;               {end PROCEDURE get203}
539
540
541 PROCEDURE Get41X ( Rec : STRING );
542 VAR
543    temp_channel    : BYTE;
544    fault           : integer;
545    temp_sat_PRN    : STRING;
546    temp_azimuth    : STRING;
547    temp_elevation  : STRING;
548
549 BEGIN
550    temp_sat_PRN := copy (Rec, 6,2);
```

```
551    temp_azimuth := copy (Rec, 9,3);
552    temp_elevation := copy (Rec, 13,2);
553    val ( copy (Rec, 4,1), temp_channel, fault);
554    IF ((fault=0) AND ((temp_channel>=1) AND (temp_channel<=max_channels)))
555    THEN BEGIN
556       {sat_geometry[temp_channel].sat_PRN := temp_sat_PRN;
557       sat_geometry[temp_channel].azimuth := temp_azimuth;
558       sat_geometry[temp_channel].elevation := temp_elevation;}
559    END;
560    IF (RecNum = ' 416') THEN writeln (fsat,Rec)
561                         ELSE write (fsat, Rec);
562 END;    { of PROCEDURE Get41X }
563
564
565 PROCEDURE unit_4200 (VAR Rec : STRING);
566 (* ------------------------------------------------------------------------ *)
567 (*                                                                          *)
568 (*  Function : convert the compressed form of data                          *)
569 (*                                                                          *)
570 (*  input  : Rec: a full line of the InputFilename                          *)
571 (*                                                                          *)
572 (*  output : AllChannelMeas : the converted raw data from rec type 2        *)
573 (*                   Navdata : the ephemeris data                           *)
574 (*                                                                          *)
575 (* ------------------------------------------------------------------------ *)
576
577 BEGIN
578    RecNum := copy (Rec,1,4);
579    IF (RecNum = '   2') THEN Get2 ( Rec, AllChannelMeas );
580    IF (RecNum = '   8') THEN Get8 ( Rec );
581    IF (RecNum = ' 200') THEN Get200 ( Rec, Navdata );
582    IF (RecNum = ' 201') THEN Get201 ( Rec, Navdata );
583    IF (RecNum = ' 202') THEN Get202 ( Rec, Navdata );
584    IF (RecNum = ' 203') THEN Get203 ( Rec, Navdata );
585    IF (RecNum = ' 411') THEN Get41X ( Rec );
586    IF (RecNum = ' 412') THEN Get41X ( Rec );
587    IF (RecNum = ' 413') THEN Get41X ( Rec );
588    IF (RecNum = ' 414') THEN Get41X ( Rec );
589    IF (RecNum = ' 415') THEN Get41X ( Rec );
590    IF (RecNum = ' 416') THEN Get41X ( Rec );
591 END;
592
593
594 BEGIN        { of UNIT }
595    CalcOnesTwo2Power ( Two2Power );
596    Last20x:=0;                          {indicates no 200..203 is received yet}
597    FOR loop:=1 TO max_sv_id DO
598       StoredHatchData[loop].NumberOfHatch:=0;   {indicates: nothing in it yet}
599 END.
600
601 (* ------------------------ END OF FILE ------------------------------- *)
```

```
(* ----------------------- FILE CALC_SAT.PAS --------------------------- *)

UNIT calc_sat;

(* ************************************************************************ *)
(*                                                                          *)
(*   Function : calculate the satellite position                            *)
(*              corrected the satellite clock                               *)
(*                                                                          *)
(* ************************************************************************ *)
INTERFACE
{$N+}

USES
   common,
   types;

PROCEDURE CorrectTOT (NoOfCh       : byte;
                      ACM          : AllChannelMeasType;
                      ArrayNavData : ArrayNavDataType;
                      VAR res      : ResultType);
(* ------------------------------------------------------------------------ *)
(*                                                                          *)
(*   INPUT :       NoOfCh : number of channel                               *)
(*                    ACM : from AllChannelMeas, which contains the         *)
(*                          converted raw satellite data                    *)
(*           ArrayNavData : array of the ephemeris data                     *)
(*   OUTPUT : res : the corrected transmission time                         *)
(*                     and the satellite position                           *)
(*                                                                          *)
(* ------------------------------------------------------------------------ *)

IMPLEMENTATION

FUNCTION EccentricAnomaly (    Mk             : float;
                               e              : float;
                           VAR IterationError : boolean ) : float;
{************************************************************************}
{ This function solves the EccentricAnomaly (Ek) by iteration.
  The equation to be solved is : Mk = Ek - e sin( Ek).
  Using Picard Iteration Ekn+1 = Mk + e sin (Ekn).
  This gives linear convergence.
  Quadratic convergence occurs using Newton Raphson:
            f(Ek)=0;         Ekn+1 = Ekn - f(Ekn)/f'(Ekn);
  This can be used if  * f'(Ek)<>0
                       * f(Ek) 3 times continious differential
  Because f(Ek)=0 for only one point it always converges.
  Newton Raphson is used, the starting point is get by picard iteration:
     Ekn = Mk + e sin (Mk),
  followed by Newton-Raphson:
     Ekn+1 = Ekn - f(Ekn)/f'(Ekn) with  * f(Ek) = Mk - Ek + e sin (Ek)  and
                                        * f'(Ek)= -1 + e cos (Ek)
  Iteration is finished when abs(Ekn+1 - Ekn) < epsilon
  Input    : Mean Anomaly Mk
             Eccentricity e
  Output   : Eccentric Anomaly Ek
  Uses     : -                                                           }
{************************************************************************}
CONST
   epsilon       = 1E-12;   {accuracy for Newton Raphson}
   MaxIterations = 10;      {maximum number of iterations}

VAR
   Ek_loc            : float;              {local value of Ek}
   deltaEk           : float;              {used for NR iteration}
   NumberOfIterations : integer;           {number of iterations}

BEGIN
   NumberOfIterations:=0;
   Ek_loc := Mk + e * sin (Mk);            {starting value of Ek, using Picard}
   REPEAT
      deltaEk := ( Mk - Ek_loc + e * sin(Ek_loc) )/( -1 + e * cos(Ek_loc) );
                                                                {f(Ek)/f'(Ek)}
      Ek_loc := Ek_loc - deltaEk;                              {Newton Raphson}
      inc(NumberOfIterations);
   UNTIL ( (abs(deltaEk) < epsilon) OR (NumberOfIterations > MaxIterations) );
   IterationError := (NumberOfIterations > MaxIterations);  {Max number of iter}
   EccentricAnomaly := Ek_loc;
END;              { of FUNCTION EccentricAnomaly }


PROCEDURE SVPosition( TimeOfTransmission : float;
                      TimeOfReception    : float;
                      VAR NavData        : NavDataType;
                      VAR PosX           : float;
                      VAR PosY           : float;
                      VAR PosZ           : float;
                      VAR IterationError : boolean);
{************************************************************************}
{ This procedure calculates the position of the satellite in ECEF coordinates
  at the (corrected) time of transmission, rotated to the time of reception
  using ephemeris data. Used is the document ICD-GPS-200.

  First the ECEF position at the time of transmission is calculated. After
  this we get the ECEF position at the time of reception by rotating this
  position. information found in 'F.J.J. Brouwer: GPS' p. 2.6.
  The value of pi used for conversion from semi-circles to radians is
  according to ICD-GPS-200 pi = 3.1 415 926 535 898
  An iterationError can occur in the EccentricAnomaly procedure.
            Input  : TimeOfTransmission
                     TimeOfReception
                     NavData
            Output : SV_ECEF
                     IterationError
            Uses   : EccentricAnomaly                                   }
{************************************************************************}
CONST
   mu        = 3.986005E+14;             {WGS84 value of the earth's universal
                                          gravitational parameter, ICD-GPS 200}
   pi_WGS    = 3.1415926535897932385;              {WGS 84 value, 19 decimals}
```

```
111    OMEGAedot = 7.2921151467E-5;    {WGS 84 value of the earth's rotation, rad/s}
112
113 VAR
114    tk,                        {time from Ephemeris Reference Epoch}
115    Axis,                      {Semi-Major Axis}
116    no,                        {computed mean motion}
117    n,                         {corrected mean motion}
118    Mk,                        {Mean Anomaly}
119    Ek,
120    vk,                        {True Anomaly}
121    vkTeller,                  {numerator for calculating Vk}
122    vkNoemer,                  {denumerator for calculating Vk}
123    Phik,                      {argument of latitude}
124    TwoPhik,                   {2 * phik}
125    SinTwoPhik,                {sin of twophik}
126    CosTwoPhik,                {cos of twophik}
127    deltauk,                   {argument of latitude correction}
128    deltark,                   {radius correction}
129    deltaik,                   {correction to inclination}
130    uk,                        {corrected argument of latitude}
131    rk,                        {corrected radius}
132    ik,                        {corrected inclination}
133    Cosik,                     {cos(ik)}
134    Sinik,                     {sin(ik)}
135    xk1,                       {Positions in orbital plane}
136    yk1,
137    cosOMEGAk,                 {cos(omegak)}
138    sinOMEGAk,                 {sin(omegak)}
139    OMEGAk          : float;   {corrected longitude of ascending node}
140    x_loc,
141    y_loc,
142    z_loc           : float;   {local ECEF position parameters}
143    Angle_rotate,
144    Sin_rotate,
145    Cos_rotate      : float;   {help parameters for rotating the ECEF coordinates}
146
147 BEGIN              { of PROCEDURE SVPosition }
148   WITH NavData DO
149   BEGIN
150     Axis := sqr( Asqrt);                                    {Semi-major Axis}
151     no   := sqrt( mu/Axis)/Axis;                  {mu is a local constant}
152     tk   := TimeOfTransmission - toe;      {Time from ephemeris reference epoch}
153       IF (tk > 302400)
154         THEN tk:= tk - 604800;
155       IF (tk < -302400)
156         THEN tk:= tk + 604800;                  {tk in range -302400..302400}
157     n    := no + deltan;                             {Corrected mean motion}
158     Mk   := Mo + n * tk;                                    {Mean anomaly}
159     Ek   := EccentricAnomaly(Mk, e, IterationError);  {Compute the true anomaly}
160                                {Iterative solution of Mk = Ek - e * sin(Ek)}
161                                                  {IterationError gets value}
162     vkTeller := sqrt(1 - sqr(e)) * sin (Ek);      {calculate numerator of vk}
163     vkNoemer := cos(Ek) - e;                    {calculate denumerator of vk}
164
165     IF abs( vkNoemer) < 1E-150                {protect againt division by zero}
166     THEN BEGIN
167       IF vkTeller >= 0
168         THEN vk:= pi_WGS * 0.5
169         ELSE vk:= pi_WGS * 1.5;
170     END
171     ELSE BEGIN                                          {range arctan [-pi/2..pi/2]}
172       vk:= arctan(vkTeller/vkNoemer);                          {range vk [0..2*pi]}
173       IF (vkNoemer < 0)                                      {2nd or 3th quadrant}
174         THEN vk:= vk + pi_WGS;
175       IF (vkTeller < 0) AND (vkNoemer > 0)                          {4th quadrant}
176         THEN vk:= vk + 2 * pi_WGS;
177     END;
178     Phik   := vk + omega;                                {Argument of lattitude}
179     TwoPhik:= 2 * Phik;
180     SinTwoPhik:= sin( TwoPhik);
181     CosTwoPhik:= cos( TwoPhik);                                {local constants}
182     deltauk:= Cus * SinTwoPhik + Cuc * CosTwoPhik;  {Argument of lattitude cor.}
183     deltark:= Crc * CosTwoPhik + Crs * SinTwoPhik;           {Radius correction}
184     deltaik:= Cic * CosTwoPhik + Cis * SinTwoPhik;   {Correction of Inclination}
185     uk     := Phik + deltauk;                   {Corrected argument of lattitude}
186     rk     := Axis * (1 - e * cos(Ek)) + deltark;             {Corrected radius}
187     ik     := io + deltaik + IDOT * tk;                  {Corrected inclination}
188     xk1    := rk * cos(uk);
189     yk1    := rk * sin(uk);                             {Positions in orbital plane}
190     OMEGAk := OMEGAo + ( OMEGADOT - OMEGAedot) * tk - OMEGAedot * toe;
191                                        {correct longitude of the ascending node}
192   END;        {end of with NavData do}
193   SinOMEGAk:= sin( OMEGAk);
194   CosOMEGAk:= cos( OMEGAk);
195   Cosik    := cos( ik);
196   Sinik    := sin( ik);                                              {local variables}
197
198   x_loc := xk1 * cosOMEGAk - yk1 * cosik * sinOMEGAk;
199   y_loc := xk1 * sinOMEGAk + yk1 * cosik * cosOMEGAk;
200   z_loc := yk1 * sinik;    {Earth-fixed coordinates (ECEF) at time of transmis.}
201   Angle_rotate:= (TimeOfReception - TimeOfTransmission)*OMEGAedot;
202                                                       {rotation angle to the left}
203   Cos_rotate:= cos(Angle_rotate);
204   Sin_rotate:= sin(Angle_rotate);
205   PosX := x_loc *  Cos_rotate + y_loc * Sin_rotate;
206   PosY := x_loc * -Sin_rotate + y_loc * Cos_rotate;       {Rotation to the left}
207   PosZ := z_loc;           {Earth-fixed coordinates (ECEF) at time of reception}
208 END;         { of PROCEDURE SVPosition }
209
210
211 PROCEDURE SVClockCorrection ( VAR TimeOfTransmission : float;
212                               VAR NavData            : NavDataType;
213                               VAR IterationError     : boolean);
214 {*****************************************************************************}
215 { This procedure corrects the SV time of transmission when only the
216   L1 frequency is used with:
217     * clockparameters (clockoffset, drift and drift rate) (ICD-GPS-200., p72)
218     * relativistic effect  (ICD-GPS-200., p73)
219     * L1-L2 frequency   (ICD-GPS-200., p74)
220         Input : NavData (clock & ephemeris)
```

```
221                TimeOfTransmission
222          Output: TimeOfTransmission (corrected)
223                  IterationError
224          Uses  : EccentricAnomaly                                          }
225 {*************************************************************************}
226 CONST
227     F  = -4.442807633E-10;   {s/m^.5, ICD-GPS-200, p. 73}
228     mu = 3.986005E+14;
229     {WGS 84 value of the earth's universal gravitational parameter, ICD 200 GPS}
230
231 VAR
232    {used for relativistic correction}
233    tk        :  float;          {time reference to toe}
234    Axis,                        {Semi-Major Axis}
235    no,                          {computed mean motion}
236    n,                           {corrected mean motion}
237    Mk,                          {Mean Anomaly}
238    Ek        : float;           {EccentricAnomaly}
239    DELTAtr   : float;           {relativistische correctie}
240    {used for normal correction}
241    ToT,                     {SV prn code phase at message transmission time [s]}
242                                 {help variable}
243    DELTAtsv  : float;           {SV prn code phase time offset [s]}
244    DELTAl1   : float;           {L1-L2 correction}
245
246 BEGIN
247    ToT:=TimeOfTransmission;                          {copy Time of transmission}
248
249   { C O R R E C T   F O R   C L O C K P A R A M E T E R S }
250      WITH NavData DO
251      BEGIN
252           dt:= ToT - toc;
253               IF (dt > 302400)
254                   THEN dt:= dt - 604800;
255               IF (dt < -302400)
256                   THEN dt:= dt + 604800;          {dt in range -302400..302400}
257           DELTAtsv:= af0 + af1 * dt + af2 * sqr( dt);
258           Tot := Tot - DELTAtsv;
259                        {Time of Transmission corrected with clockparameters}
260
261   { C O R R E C T   F O R   R E L A T I V I S T I C   E F F E C T }
262           Axis := sqr( Asqrt);                                  {Semi-major Axis}
263           no   := sqrt(mu/Axis)/Axis;                 {mu is a local constant}
264           tk   := ToT - toe;              {Time from ephemeris reference epoch}
265                  if (tk > 302400)
266                      then tk:= tk - 604800;
267                  if (tk < -302400)
268                      then tk:= tk + 604800;       {tk in range -302400..302400}
269           n    := no + deltan;                        {Corrected mean motion}
270           Mk   := Mo + n * tk;                                {Mean anomaly}
271           Ek   := EccentricAnomaly(Mk, e, IterationError);
272                                                  {Compute the true anomaly}
273                                  {Iterative solution of Mk = Ek - e * sin(Ek)}
274                                                 {IterationError gets value}
275           DELTAtr := F * e * Asqrt * sin(Ek);          {relativistic correction}
276
277   { C O R R E C T   F O R   L1 - L2   F R E Q U E N C Y }
278            DELTAl1:=Tgd;
279       end;                {with NavData do}
280    TimeOfTransmission := TimeOfTransmission - DELTAtsv - DELTAtr + DELTAl1;
281                                                  {corrected time of transmission}
282 END;           { of PROCEDURE SVClockCorrection }
283
284
285 PROCEDURE CorrectTOT (NoOfCh       : byte;
286                       ACM          : AllChannelMeasType;
287                       ArrayNavData : ArrayNavDataType;
288                       VAR res      : ResultType);
289 {*************************************************************************}
290 { INPUT : NoOfCh: aantal satellieten in data.
291           UT    : unc-TOR, unc-TOT's for every satellite
292           ArrayNavData    : Navigation data
293   OUTPUT: res   : Resultaat: satellite position, Pseudorange error
294   USES  : ClockError, SpeedLight, RefX, RefY, RefZ
295   * Correct time of tranmission
296   * Calculate sat. position using an estimated time of reception (ClockError!)
297   * Calculate true range using reference position
298   * Calculate Pseudorange error: receivertime and corrected TOT
299   * Calculate error in estimated time of reception. If this error is to large,
300     repeat calculations
301   * Houdt ClockError bij.                                                  }
302 {*************************************************************************}
303 VAR
304    i               : byte;
305    IterationError  : boolean;
306    AbleToCalculate : boolean;
307    TempTime        : float;
308    TempPRN         : byte;
309    locX,
310    locY,
311    locZ            : float;
312    TrueRange,
313    tempPR,
314    tempPRE,
315    TotalPRE        : float;    {Total of PRE's in res}
316    NumberOfPRE     : integer;  {Number of PRE's in res}
317    TempInc         : float;
318    user_sec        : longint;
319
320 BEGIN           { of PROCEDURE CorrectTOT }
321   NumberOfPRE := 0;
322   TotalPRE    := 0;
323   res.UncTOR  := ACM.user_ms;
324   res.EstTOR  := ACM.user_ms +
325                  ClockError.ClockOffsetSeconds +
326                  ClockError.ClockDrift *
327                     GPStimeIncrement(ClockError.LastUncTor,ACM.user_ms);
328         {Estimated time of reception}
329
330   FOR i:=1 TO 6 DO
```

```
331  BEGIN
332    TempPRN         := ACM.channel[i].PRN;
333    AbleToCalculate := (TempPRN<>0) AND (ArrayNavData[TempPRN].PRN = (tempPRN));
334
335    IF AbleToCalculate                                {Nav data and measurement}
336    THEN BEGIN
337      res.channel[i].PRN       := TempPRN;
338      res.channel[i].NoOfHatch := ACM.channel[i].NoOfHatch;
339      TempTime                 := ACM.channel[i].time_of_transmission;
340
341      SVClockCorrection(TempTime, ArrayNavData[TempPRN], IterationError);
342      IF IterationError
343        THEN writeln('IterationError in Satellite Clock correction');
344
345      SVPosition
346      (TempTime,res.EstTOR,ArrayNavData[TempPRN],locX,locY,locZ,IterationError);
347      IF IterationError
348        THEN writeln('IterationError in Satellite Position calculation');
349
350      TrueRange := sqrt( sqr(locX-refX) + sqr(locY-refY) + sqr(locZ-refZ));
351      tempPR    := (res.UncTOR - TempTime) * light_speed;
352      tempPRE   := TrueRange - tempPR;
353      TotalPRE := TotalPRE + tempPRE;
354      inc(NumberOfPRE);
355
356      WITH res.channel[i] DO
357      BEGIN
358         CorTOT        := TempTime;
359         PR            := tempPR;
360         sv_position.x := LocX;
361         sv_position.y := LocY;
362         sv_position.z := LocZ;
363         PRE           := tempPRE;
364      END;
365    END                 {IF AbleToCalculate THEN}
366    ELSE WITH res.channel[i] DO         {no measurement or nav data available}
367      BEGIN
368        PRN           := 0;
369        NoOfHatch     := 0;
370        CorTOT        := 0;
371        PR            := 0;
372        sv_position.x := 0;
373        sv_position.y := 0;
374        sv_position.z := 0;
375        PRE           := 0;
376      END;
377  END;          {FOR i:=1 TO NoOfCh DO}
378
379  IF (NumberOfPRE<>0)
380  THEN BEGIN
381    res.AveragePRE := TotalPRE/NumberOfPRE;
382    TotalPRE:=(TotalPRE/NumberOfPRE)/light_speed;    {clock offset in seconds}
383    IF (ClockError.NoOfMeas=0)                            {first measurement}
384    THEN BEGIN
385      ClockError.LastUncTOR         := res.UncTOR;
386      ClockError.NoOfMeas           := 1;
387      ClockError.ClockOffsetSeconds := TotalPRE;
388      ClockError.ClockDrift         := 0;
389    END
390    ELSE BEGIN
391      TempInc := GPStimeIncrement (ClockError.LastUncTOR, res.UncTOR);
392      IF (abs(TempInc) > 1E-4)
393         THEN ClockError.ClockDrift :=
394              (TotalPRE - ClockError.ClockOffsetSeconds) / TempInc;
395      ClockError.LastUncTOR := ACM.user_ms;
396      IF (ClockError.NoOfMeas<100) THEN inc(ClockError.NoOfMeas);
397      ClockError.ClockOffsetSeconds := TotalPRE;
398    END;
399
400    IF abs (GPStimeIncrement (res.EstTOR, res.UncTOR + TotalPRE) ) > 2E-7
401                                                                    {60 m}
402    THEN CorrectTOT (NoOfCh, ACM, ArrayNavData, res);            {recursive}
403    res.EstTOR := ACM.user_ms + TotalPRE;
404                                       {updated estimate of time of reception}
405  END          {end of NumberOfPRE<>0}
406
407  ELSE res.AveragePRE:=0;
408
409 END;      { of PROCEDURE CorrectTOT }
410
411
412 BEGIN
413    ClockError.LastUncTOR         := 0;
414    ClockError.NoOfMeas           := 0;
415    ClockError.ClockOffsetSeconds := 0;
416    ClockError.ClockDrift         := 0;
417 END.
418
419 (* ------------------------ END OF FILE ------------------------------- *)
```

```
(* ---------------------- FILE CALC_POS.PAS --------------------------- *)

UNIT calc_pos;

(* ************************************************************************ *)
(*                                                                          *)
(*   Function : calculate the user position in ECEF-xyz coordinates         *)
(*                                                                          *)
(* ************************************************************************ *)

INTERFACE
{$N+,E+}

USES
   types,
   matrixex;


PROCEDURE calc_position(    num_of_sv,
                            NoOfCh   : integer;
                        VAR XYZ      : ResultType;
                        VAR Rec      : ResultType;
                        VAR Q, SVpos : TNmatrix);
(* ------------------------------------------------------------------------ *)
(*   INPUT : num_of_sv : number of satellite                                *)
(*              NoOfCh : number of channel                                  *)
(*   OUTPUT : XYZ : satellite coordinates in ECEF-coordinates               *)
(*            Rec : receiver coordinates in ECEF-coordinates                *)
(* ------------------------------------------------------------------------ *)

IMPLEMENTATION


PROCEDURE least_sq_meth ( UserEst  : TNvector;
                          SVpos    : TNmatrix;
                          num_of_sv : byte;
                      VAR UserPos  : TNvector;
                      VAR InvAtA   : TNmatrix);
(* ------------------------------------------------------------------------ *)
(*  Function : Calculate the least squares                                  *)
(*  INPUT : SVpos    : 'NumOfSVs' rows(1:4) containing pseudorange, x, y, z *)
(*          UserEst :  vector x, y, z, clockerror(m)                        *)
(*  OUTPUT: UserPos                                                         *)
(* ------------------------------------------------------------------------ *)
VAR
   i, j,
   SVs          : byte;
   delta,
   range        : double;
   H , Ht ,
   delta_x,
   delta_y,
   delta_z      : TNmatrix;
   Error        : byte;

CONST
   epsilon = 0.01;

BEGIN
   REPEAT
      FOR SVs:=1 TO num_of_sv DO
      BEGIN
         range          := sqrt( sqr(SVpos[SVs,1]-UserEst[1]) +
                                 sqr(SVpos[SVs,2]-UserEst[2]) +
                                 sqr(SVpos[SVs,3]-UserEst[3]));
         delta_z[SVs,1] := SVpos[SVs,4] - range - UserEst[4];          {z-zpred}
         H[SVs,1]       := (UserEst[1] - SVpos[SVs,1]) / range;
         H[SVs,2]       := (UserEst[2] - SVpos[SVs,2]) / range;
         H[SVs,3]       := (UserEst[3] - SVpos[SVs,3]) / range;
         H[SVs,4]       := 1;
      END;

      Transpose(num_of_sv, 4, H, Ht );                                  {Ht = H'}
      Matrix_Mult(4, num_of_sv, 4, Ht, H, Q);                         {Q = H'* H}
      Inverse(4, Q, InvAtA, Error);                                  {Q = inv(Q)}
      Matrix_Mult(4, 4, num_of_sv, InvAtA, Ht, delta_x);      {delta_x = Q * Ht}
      Matrix_Mult(4, num_of_sv, 1, delta_x, delta_z, delta_y);

      FOR i:=1 TO 4 DO
         UserEst[i] := UserEst[i] + delta_y[i,1];

      delta := sqrt ( sqr(delta_y[1,1]) + sqr(delta_y[2,1]) +
                      sqr(delta_y[3,1]) + sqr(delta_y[4,1]) );

   UNTIL (delta < epsilon);

   FOR j:=1 TO 4 DO
      UserPos[j]:=UserEst[j];
END;               { of PROCEDURE LsqUserPos }


PROCEDURE calc_position (   num_of_sv,
                            NoOfCh   : integer;
                        VAR XYZ      : ResultType;
                        VAR Rec      : ResultType;
                        VAR Q, SVpos : TNmatrix);

VAR
   i, j, k          : byte;
   TempPRN          : byte;
   AbleToCalculate : boolean;

BEGIN                    { of PROCEDURE calc_position }
   num_of_sv  := 0;
   Rec.UncTOR := XYZ.UncTOR;
   Rec.EstTOR := XYZ.EstTOR;
   FOR i:=1 TO NoOfCh DO
   BEGIN
      TempPRN          := XYZ.channel[i].PRN;
      AbleToCalculate := (TempPRN<>0);
```

```
111       IF AbleToCalculate
112       THEN BEGIN
113          inc(num_of_sv);
114                                    {store the satellite coordinates to SVpos}
115          SVpos[num_of_sv,1] := XYZ.channel[i].sv_position.x;     {x-coordinates}
116          SVpos[num_of_sv,2] := XYZ.channel[i].sv_position.y;     {y-coordinates}
117          SVpos[num_of_sv,3] := XYZ.channel[i].sv_position.z;     {z-coordinates}
118          SVpos[num_of_sv,4] := XYZ.channel[i].PR                  {pseudo range}
119       END;
120    END;
121
122    IF num_of_sv > 3
123    THEN BEGIN
124       FOR j:=1 TO 4 DO  UserEst[j] := 0;
125       least_sq_meth (UserEst, SVpos, num_of_sv, UserPos, Q);
126                                          {Least Square Methods - calculation}
127       FOR k:=1 TO 4 DO  Rec.Position[k] := UserPos[k];
128    END
129    ELSE Clean_matrix(4, 4, Q);
130 END;            { of PROCEDURE calc_position }
131
132
133 END.
134
135 (* ------------------------ END OF FILE -------------------------------- *)
```

```
(* ---------------------- FILE MATRIXEX.PAS --------------------------- *)

UNIT matrixex;

(* ************************************************************************ *)
(*                                                                          *)
(*   Function : contains procedures for matrix calculation                  *)
(*                                                                          *)
(* ************************************************************************ *)
{$I Float.inc}

INTERFACE

USES
   types;

{$IFOPT N+}
TYPE
  Float = Double; { 8 byte real, requires 8087 math chip }
CONST
  TNNearlyZero = 1E-15;

{$ELSE}
TYPE
  Float = real;   { 6 byte real, no math chip required }
CONST
  TNNearlyZero = 1E-07;
{$ENDIF}


PROCEDURE Transpose(    TotalOfRows,
                        TotalOfCols : BYTE;
                        Matrix      : TNmatrix;
                    VAR SolMatrix   : TNmatrix );

PROCEDURE Matrix_Mult (    TotalOfRowsM1,
                           TotalOfColsM1,
                           TotalOfColsM2 : BYTE;
                           Matrix1,
                           Matrix2       : TNMATRIX;
                       VAR SolMatrix     : TNMATRIX );

PROCEDURE Clean_Matrix (    TotalOfRows,
                            TotalOfCols : BYTE;
                        VAR Matrix      : TNmatrix);

PROCEDURE Inverse(    Dimen : integer;
                      Data  : TNmatrix;
                  VAR Inv   : TNmatrix;
                  VAR Error : byte);

IMPLEMENTATION

PROCEDURE Transpose (    TotalOfRows,
                         TotalOfCols : BYTE;
                         Matrix      : TNmatrix;
                     VAR SolMatrix   : TNmatrix );
(* ------------------------------------------------------------------------ *)
(*  This procedure calculates SolMatrix (SolutionMatrix)                    *)
(*                            = The transpose of Matrix.                    *)
(*  Input : TotalOfRows = Integer with Total of rows in Matrix              *)
(*          TotalOfCols = Integer with Total of columns in Matrix           *)
(*          Matrix      = TNMatrix to be processed                          *)
(*  Output: SolMatrix   = TNMatrix is the result                            *)
(* ------------------------------------------------------------------------ *)
VAR i,j : byte;

BEGIN
  FOR i:=1 TO TotalOfRows DO
  BEGIN
    FOR j:=1 TO TotalOfCols DO
       SolMatrix[j,i] := Matrix[i,j];
  END;
END;    { of PROCEDURE Transpose }


PROCEDURE Matrix_Mult (    TotalOfRowsM1,
                           TotalOfColsM1,
                           TotalOfColsM2 : BYTE;
                           Matrix1,
                           Matrix2       : TNMATRIX;
                       VAR SolMatrix     : TNMATRIX );
(* ------------------------------------------------------------------------ *)
(*  This procedure calculates SolMatrix (SolutionMatrix) = Matrix1*Matrix2. *)
(* Input : TotalOfRowsM1 = Integer with the Total of rows in Matrix1        *)
(*         TotalOfColsM1 = Integer with Total of columns in Matrix1         *)
(*         TotalOfColsM2 = Integer with Total of columns in Matrix2         *)
(*         Matrix1       = TNMatrix with first argument                     *)
(*         Matrix2       = TNMatrix with second argument                    *)
(* Output: SolMatrix     = TNMatrix with result                             *)
(* ------------------------------------------------------------------------ *)
VAR i,j : byte;

FUNCTION RowColumnMult ( LineLengthM1,
                         RowM1   : byte;
                         Matrix1 : TNmatrix;
                         ColM2   : BYTE;
                         Matrix2 : TNMATRIX ) : float;
(* ------------------------------------------------------------------------ *)
(*  This function calculates the product of a row in Matrix1 (RowM1) and    *)
(*   a coloumn in Matrix2 (ColM2).                                          *)
(*  Input : LineLenghtM1 = Integer with The length of a line in Matrix1     *)
(*                         (Total of columns in Matrix1)                    *)
(*         RowM1         = Integer with The rownumber of Matrix1           *)
(*         ColM2         = Integer with The columnnumber of Matrix2        *)
(*         Matrix1       = TNMatrix of first argument                      *)
(*         Matrix2       = TNMatrix of second argument                     *)
(*  Output: RowColumnMult= float with the result of the row-column         *)
(*                          multiplication.                                 *)
(* ------------------------------------------------------------------------ *)
```

```
111 VAR term : byte;
112     Sum  : float;
113
114 BEGIN
115    Sum:=0;
116    FOR term:=1 TO LineLengthM1 DO
117       Sum := Sum + Matrix1[RowM1,term] * Matrix2[term,ColM2];
118    RowColumnMult := Sum;
119 END;    { of FUNCTION RowColumnMult }
120
121
122 BEGIN   { of PROCEDURE Matrix_Mult }
123   FOR i:=1 TO TotalOfRowsM1 DO
124   BEGIN
125     FOR j:=1 TO TotalOfColsM2 DO
126       SolMatrix[i,j] := RowColumnMult (TotalOfColsM1, i, Matrix1, j, Matrix2);
127   END;
128 END;    { of PROCEDURE Matrix_Mult }
129
130
131 PROCEDURE Clean_Matrix (    TotalOfRows,
132                             TotalOfCols : BYTE;
133                         VAR Matrix      : TNMATRIX );
134 (* ------------------------------------------------------------------------ *)
135 (*  This procedure sets all coefficients in Matrix to zero.                 *)
136 (*  Input : TotalOfRows = Integer with Total of rows in Matrix              *)
137 (*          TotalOfCols = Integer with Total of columns in Matrix          *)
138 (*  Output: Matrix          = TNMatrix with all zeros.                      *)
139 (* ------------------------------------------------------------------------ *)
140 VAR i,j : byte;
141
142 BEGIN
143   FOR i:=1 TO TotalOfRows DO
144   BEGIN
145     FOR j:=1 TO TotalOfCols DO Matrix[i,j]:=0;
146   END;
147 END;  { of PROCEDURE Clean_Matrix }
148
149
150 procedure Inverse(Dimen : integer;
151                   Data   : TNmatrix;
152               var Inv    : TNmatrix;
153               var Error  : byte);
154
155 procedure Initial(Dimen : integer;
156               var Data  : TNmatrix;
157               var Inv   : TNmatrix;
158               var Error : byte);
159
160 {-------------------------------------------------------}
161 {- Input: Dimen, Data                                   -}
162 {- Output: Inv, Error                                   -}
163 {-                                                      -}
164 {- This procedure test for errors in the value of Dimen -}
165 {-------------------------------------------------------}
166
167 var
168   Row : integer;
169
170 begin
171   Error := 0;
172   if Dimen < 1 then
173     Error := 1
174   else
175     begin
176       { First make the inverse-to-be the identity matrix }
177       FillChar(Inv, SizeOf(Inv), 0);
178       for Row := 1 to Dimen do
179         Inv[Row, Row] := 1;
180       if Dimen = 1 then
181         if ABS(Data[1, 1]) < TNNearlyZero then
182           Error := 2   { Singular matrix }
183         else
184           Inv[1, 1] := 1 / Data[1, 1];
185     end;
186 end; { procedure Initial }
187
188 procedure ERODiv(Divisor : Float;
189                  Dimen   : integer;
190              var Row     : TNvector);
191
192 {-----------------------------------------------------}
193 {- Input: Divisor, Dimen, Row                         -}
194 {-                                                    -}
195 {- elementary row operation - dividing by a constant -}
196 {-----------------------------------------------------}
197
198 var
199   Term : integer;
200
201 begin
202   for Term := 1 to Dimen do
203     Row[Term] := Row[Term] / Divisor;
204 end; { procedure ERODiv }
205
206 procedure EROswitch(var Row1 : TNvector;
207                     var Row2 : TNvector);
208
209 {-----------------------------------------------}
210 {- Input: Row1, Row2                           -}
211 {- Output: Row1, Row2                          -}
212 {-                                             -}
213 {- Elementary row operation - switching two rows -}
214 {-----------------------------------------------}
215
216 var
217   DummyRow : TNvector;
218
219 begin
220   DummyRow := Row1;
```

```
  Row1 := Row2;
  Row2 := DummyRow;
end; { procedure EROswitch }

procedure EROmultAdd(Multiplier   : Float;
                     Dimen        : integer;
                 var ReferenceRow : TNvector;
                 var ChangingRow  : TNvector);

{-------------------------------------------------------}
{- Input: Multiplier, Dimen, ReferenceRow, ChangingRow  -}
{- Output: ChangingRow                                  -}
{-                                                      -}
{- Row operation - adding a multiple of one row to another -}
{-------------------------------------------------------}

var
  Term : integer;

begin
  for Term := 1 to Dimen do
    ChangingRow[Term] := ChangingRow[Term] + Multiplier*ReferenceRow[Term];
end; { procedure EROmultAdd }


procedure Inver(Dimen : integer;
            var Data  : TNmatrix;
            var Inv   : TNmatrix;
            var Error : byte);

{-------------------------------------------------------}
{- Input: Dimen, Data                                  -}
{- Output: Inv, Error                                  -}
{-                                                     -}
{- This procedure computes the inverse of the matrix Data -}
{- and stores it in the matrix Inv.  If the matrix Data  -}
{- is singular, then Error = 2 is returned.            -}
{-------------------------------------------------------}

var
  Divisor, Multiplier : Float;
  Row, ReferenceRow : integer;

procedure Pivot(Dimen        : integer;
                ReferenceRow : integer;
            var Data         : TNmatrix;
            var Inv          : TNmatrix;
            var Error        : byte);

{-------------------------------------------------------}
{- Input: Dimen, ReferenceRow, Data, Inv                -}
{- Output: Data, Inv, Error                             -}
{-                                                      -}
{- This procedure searches the ReferenceRow column of   -}
{- the Data matrix for the first non-zero element below -}
{- the diagonal. If it finds one, then the procedure    -}
{- switches rows so that the non-zero element is on the -}
{- diagonal. This same operation is applied to the Inv  -}
{- matrix. If no non-zero element exists in a column, the -}
{- matrix is singular and no inverse exists.            -}
{-------------------------------------------------------}

var
  NewRow : integer;

begin
  Error := 2;  { No inverse exists }
  NewRow := ReferenceRow;
  while (Error > 0) and (NewRow < Dimen) do
  { Try to find a       }
  { row with a non-zero }
  { diagonal element    }
  begin
    NewRow := Succ(NewRow);
    if ABS(Data[NewRow, ReferenceRow]) > TNNearlyZero then
    begin
      EROswitch(Data[NewRow], Data[ReferenceRow]);
      { Switch these two rows }
      EROswitch(Inv[NewRow], Inv[ReferenceRow]);
      Error := 0;
    end;
  end; { while }
end; { procedure Pivot }

begin { procedure Inver }
  { Make Data matrix upper triangular }
  ReferenceRow := 0;
  while (Error = 0) and (ReferenceRow < Dimen) do
  begin
    ReferenceRow := Succ(ReferenceRow);
    { Check to see if the diagonal element is zero }
    if ABS(Data[ReferenceRow, ReferenceRow]) < TNNearlyZero then
      Pivot(Dimen, ReferenceRow, Data, Inv, Error);
    if Error = 0 then
    begin
      Divisor := Data[ReferenceRow, ReferenceRow];
      EROdiv(Divisor, Dimen, Data[ReferenceRow]);
      EROdiv(Divisor, Dimen, Inv[ReferenceRow]);
      for Row := 1 to Dimen do
        { Make the ReferenceRow element of this row zero }
        if (Row <> ReferenceRow) and
           (ABS(Data[Row, ReferenceRow]) > TNNearlyZero) then
        begin
          Multiplier := -Data[Row, ReferenceRow] /
                        Data[ReferenceRow, ReferenceRow];
          EROmultAdd(Multiplier, Dimen, Data[ReferenceRow], Data[Row]);
          EROmultAdd(Multiplier, Dimen, Inv[ReferenceRow], Inv[Row]);
        end;
    end;
  end;
```

```
331 end; { procedure Inver }
332
333 begin { procedure Inverse }
334   Initial(Dimen, Data, Inv, Error);
335   if Dimen > 1 then
336     Inver(Dimen, Data, Inv, Error);
337 end; { procedure Inverse }
338
339
340 END.
341
342 (* ------------------------ END OF FILE ---------------------------------- *)
```

```
(* ------------------------ FILE INPUT.PAS ----------------------------- *)

UNIT input;

(* ************************************************************************ *)
(*                                                                          *)
(*   Function :                                                             *)
(*       - procedures which is used as the interface to the screen          *)
(*       - needed for the user to put the input and output file name        *)
(*                                                                          *)
(* ************************************************************************ *)

INTERFACE
{$N+}

USES
   types,
   crt;

PROCEDURE begin_user_position;
PROCEDURE output_of_magnavox (VAR input_file_name  : STRING);
PROCEDURE input_reference_position (VAR input_file_name : STRING;
                                   VAR output_file_name : STRING);
PROCEDURE input_rec_position (VAR input_file_name  : STRING;
                              VAR output_file_name : STRING;
                              VAR chan_in_input    : byte;
                              VAR state            : CHAR);


IMPLEMENTATION

USES
   common;


PROCEDURE begin_user_position;
BEGIN
  CLRSCR;
  writeln;
  writeln(' In this part, the user position is calculated.                ');
  writeln(' The diagram which is used to calculate the user position is   ');
  writeln(' given in Figure 4.3 in chapter 4.                             ');
  writeln;
  writeln;
  writeln(' Press any key to start the program                            ');
  REPEAT
  UNTIL ReadKey <> '';
END;


PROCEDURE output_of_magnavox (VAR input_file_name : STRING);
{ ---------------------------------------------------------------------- }
{   Function : screen information to convert raw data item               }
{   Input : input_file_name : name of input file name which contains     }
{                             the compressed form of raw data file       }
{ ---------------------------------------------------------------------- }
VAR
   ts        : STRING;
   ErrorCode : integer;
   ch        : char;

BEGIN
  REPEAT
   CLRSCR;
   writeln;
   write(' Input file name (8 characters, .dat) : ');
   input_file_name := ReadFromKeyBoard(8,false,'port1');
   writeln;
   write(' Output file name  (*.utt)            : ');
   output_file_name:=ReadFromKeyBoard(8,false,input_file_name);
   writeln;
   write('Hatch filtering, (N-1/N*old + 1/N*new, 1..n)         : ');
   ts:=ReadFromKeyBoard(8,true,'100');
   val(ts,HatchFil,ErrorCode);
   writeln;
   write('Measurements missing before lock out (1..n)          : ');
   ts:=ReadFromKeyBoard(8,true,'3');
   val(ts,MeasMissing,ErrorCode);
   writeln;
   write('Start at measurement number (-1: start immediately)  : ');
   ts:=ReadFromKeyBoard(8,true,'-1');
   val(ts,start_meas_no,ErrorCode);
   writeln;
   write('End at measurement number (999999: till end of file) : ');
   ts:=ReadFromKeyBoard(8,true,'999999');
   val(ts,end_meas_no,ErrorCode);
   writeln;
   writeln;
   writeln (' Record Type 2       : ', input_file_name,'.utt ');
   writeln (' Record Type 8       : ', input_file_name,'.pos ');
   writeln (' Record Type 200-203 : ', input_file_name,'.X ');
   writeln (' Record Type 308     : ', input_file_name,'.nav ');
   writeln (' Record Type 411-416 : ', input_file_name,'.sat ');
   writeln;
   writeln (' The ',input_file_name,'.utt consist of :');
   writeln (' The user time and for each channel consist of receiver time, ');
   writeln (' PRN, the following choises and Hatchfiltering.               ');
   writeln;
   writeln (' The choises in the ',input_file_name,'.utt file are:         ');
   write (' The corrected time of transmission : (y/n)  ');
   time_corrected      := (ReadFromKeyBoard(3,false,'y')='y');
   writeln;
   write (' Pseudo Range              : (y/n)  ');
   pr_state            := (ReadFromKeyBoard(3,false,'n')='y');
   writeln;
   write (' Integrated carrier phase : (y/n)  ');
   carrier_phase_state := (ReadFromKeyBoard(3,false,'n')='y');
   writeln;
   write (' Code phase                : (y/n)  ');
   code_phase_state    := (ReadFromKeyBoard(3,false,'n')='y');
```

```
111    writeln;
112
113    WHILE KeyPressed DO ch:=ReadKey;        {Remove all character from buffer}
114    writeln;
115    writeln('OK? (y/n) ');
116    REPEAT
117       ch:=ReadKey;
118    UNTIL (ch IN ['y','Y','n','N']);
119   UNTIL (ch IN ['y','Y']);
120 END;          { of PROCEDURE output_of_magnavox }
121
122
123
124 PROCEDURE input_reference_position (VAR input_file_name  : STRING;
125                                     VAR output_file_name : STRING);
126 VAR
127    TempString,
128    ts                   : STRING;
129    ReferencePositionNo : byte;
130    ref_file             : text;
131    RegelNo              : longint;
132    ErrorCode            : integer;
133
134 BEGIN
135    writeln;
136    writeln;
137    write(' Input file name (8 characters, .dat) : ');
138    input_file_name := ReadFromKeyBoard(8,false,'port1');
139    writeln;
140    write('Output file name with receiver position (*.rec)      : ');
141    output_file_name:=ReadFromKeyBoard(8,false,'ecef');
142    writeln;
143    writeln('Input file Ref. Positions:  Ref_Pos.set                    ');
144    writeln('Pseudorange Errors       : `output_file_name`.pre          ');
145    writeln;
146    writeln('  A file containing the pseudorange errors.                ');
147    writeln('  (MeasNo, EstTOR, (PRN, PRE, Hatch) * n, AveragePRE       ');
148    assign(ref_file,'Ref_Pos.set');
149    reset(ref_file);
150    RegelNo :=1;
151    ReferencePositionNo:=1;
152    writeln;
153    writeln('Reference Positions:');
154    WHILE (NOT eof(ref_file)) DO
155    BEGIN
156    IF RegelNo=1 THEN
157       BEGIN
158       readln(ref_file,TempString);
159       IF TempString<>'' THEN
160        BEGIN
161          writeln( ReferencePositionNo:3,' : ',TempString);
162          inc(ReferencePositionNo);
163        END;
164       RegelNo:=-3;
165       END
166       ELSE readln(ref_file);
167       inc(RegelNo);
168    END;
169    write('Choose Reference position          :  ');
170    ts:=ReadFromKeyBoard(8,true,'1');
171    val (ts, ReferencePositionNo, ErrorCode);
172    reset(ref_file);
173    FOR RegelNo:=1 TO (ReferencePositionNo-1)*4+1 DO readln(ref_file);
174    readln(ref_file,refX);
175    readln(ref_file,refY);
176    readln(ref_file,refZ);
177    close(ref_file);
178 END;
179
180
181 PROCEDURE input_rec_position (VAR input_file_name  : STRING;
182                               VAR output_file_name : STRING;
183                               VAR chan_in_input    : byte;
184                               VAR state            : CHAR);
185 { ------------------------------------------------------------------------ }
186 {   Function : screen information of the user position calculation item     }
187 {   Input :  input_file_name : input file name                              }
188 {           output_file_name : output file name                             }
189 {              chan_in_input : number of channel                            }
190
191 { ------------------------------------------------------------------------ }
192 VAR
193   ch        : char;
194   ts        : STRING;
195   ErrorCode : integer;
196
197 BEGIN
198   REPEAT
199     CLRSCR;
200     writeln('                    POSITION CALCULATION                    ');
201     writeln;
202     writeln('This program calculates the position of the receiver in ECEF- ');
203     writeln('coordinates.                                                  ');
204     writeln('Rawdata as input come from MX4200D satellite receiver.        ');
205     writeln;
206     write('Input file name (8 characters, *.dat)              : ');
207     input_file_name:=ReadFromKeyBoard(8,false,'port1');
208     writeln;
209     write('Output file name with converted data (*.utt)       : ');
210     input_file_name:=ReadFromKeyBoard(8,false,'port1');
211     writeln;
212     write('Output file name with receiver position (*.rec)    : ');
213     output_file_name:=ReadFromKeyBoard(8,false,'ecef');
214     writeln;
215     write('Number of Channels                                 : ');
216     ts:=ReadFromKeyBoard(8,true,'6');
217     val(ts,chan_in_input,ErrorCode);
218     writeln;
219     write('Hatch filtering, (N-1/N*old + 1/N*new, 1..n)       : ');
220     ts:=ReadFromKeyBoard(8,true,'100');
```

```
221     val(ts,HatchFil,ErrorCode);
222     writeln;
223     write('Measurements missing before lock out (1..n)          : ');
224     ts:=ReadFromKeyBoard(8,true,'3');
225     val(ts,MeasMissing,ErrorCode);
226     writeln;
227     write('Start at measurement number (-1: start immediately)  : ');
228     ts:=ReadFromKeyBoard(8,true,'-1');
229     val(ts,start_meas_no,ErrorCode);
230     writeln;
231     write('End at measurement number (999999: till end of file) : ');
232     ts:=ReadFromKeyBoard(8,true,'999999');
233     val(ts,end_meas_no,ErrorCode);
234     writeln;
235
236     WHILE KeyPressed DO ch:=ReadKey;        {Remove all character from buffer}
237     writeln;
238     writeln('OK? (y/n) ');
239     REPEAT
240       ch:=ReadKey;
241     UNTIL (ch IN ['y','Y','n','N']);
242   UNTIL (ch IN ['y','Y']);
243 END;           { of PROCEDURE input_rec_position }
244
245
246 END.
247
248 (* ------------------------ END OF FILE -------------------------------- *)
```

```
(* ------------------------ FILE FILES.PAS ------------------------------ *)

UNIT files;

INTERFACE
{$N+}

USES
   crt,
   types;

(* ----- THE FILES OPERATION FOR THE NAVIGATION DATA ----- *)

PROCEDURE reset_navdata_files (    file_name   : STRING;
                               VAR NavDataFile : NavDataFileType );
PROCEDURE write_navdata_to_file (    SVnum          : byte;
                                     LastMeasNumber : longint;
                                VAR NavDataFile     : NavDataFileType;
                                VAR ND              : NavDataType );


(* ----- THE FILES OPERATION FOR THE RAW DATA ----- *)

PROCEDURE open_files_for_magnavox (input_file  : STRING;
                                   output_file : STRING );
PROCEDURE reopen_output_files;
PROCEDURE write_rawdata_to_file (    ACM  : AllChannelMeasType;
                                 VAR futt : text);
PROCEDURE close_files_for_magnavox;


(* ----- THE FILES OPERATION FOR SATELLITE POSITION CALCULATION ----- *)

{PROCEDURE read_data_from_utt_file (    NoOfCh         : integer;
                                   VAR AllChannelMeas : AllChannelMeasType);}


(* ----- THE FILES OPERATION FOR DIFFERENTIAL CORRECTION        ----- *)
PROCEDURE open_files_diff_correction (input_file  : STRING;
                                      output_file : STRING);
PROCEDURE write_to_file_pre (   result : ResultType;
                             VAR fpre  : text);
PROCEDURE close_files_diff_correction;


(* ----- THE FILES OPERATION FOR THE POSITION CALCULATION (gps) ----- *)

PROCEDURE open_files_rec_position (input_file  : STRING;
                                   output_file : STRING);
PROCEDURE write_to_file_rec (    NumOfSVs : byte;
                             VAR Rec      : ResultType);
PROCEDURE close_files_rec_position;


IMPLEMENTATION


(* ----- THE FILES OPERATION FOR THE NAVIGATION DATA ----- *)

PROCEDURE reset_navdata_files (    file_name   : STRING;
                               VAR NavDataFile : NavDataFileType);
VAR
   i            : byte;
   NumberString : STRING;

BEGIN
   FOR i:=1 TO max_sv_id DO
   BEGIN
      str(i,NumberString);
      IF (i < 10) THEN NumberString := '0'+ NumberString;
      NumberString := file_name + '.' + NumberString;
      assign(NavDataFile[i],NumberString);
      rewrite(NavDataFile[i]);
      close(NavDataFile[i]);
   END;
END;                { of PROCEDURE reset_navdata_files }


PROCEDURE write_navdata_to_file (    SVnum          : byte;
                                     LastMeasNumber : longint;
                                VAR NavDataFile     : NavDataFileType;
                                VAR ND              : NavDataType);
BEGIN
   append(NavDataFile[SVnum]);
   WITH ND DO
   BEGIN
      writeln(NavDataFile[SVnum],IODC:25);
      writeln(NavDataFile[SVnum],IODC_MSB:25);
      writeln(NavDataFile[SVnum],toc:25);
      writeln(NavDataFile[SVnum],Tgd:25);
      writeln(NavDataFile[SVnum],af2:25);
      writeln(NavDataFile[SVnum],af1:25);
      writeln(NavDataFile[SVnum],af0:25);
      writeln(NavDataFile[SVnum],Accuracy:25);
      writeln(NavDataFile[SVnum],Health:25);
      writeln(NavDataFile[SVnum],WeekNo:25);
      writeln(NavDataFile[SVnum],IODE2:25);
      writeln(NavDataFile[SVnum],Crs:25);
      writeln(NavDataFile[SVnum],deltan:25);
      writeln(NavDataFile[SVnum],Mo:25);
      writeln(NavDataFile[SVnum],Cuc:25);
      writeln(NavDataFile[SVnum],e:25);
      writeln(NavDataFile[SVnum],Cus:25);
      writeln(NavDataFile[SVnum],Asqrt:25);
      writeln(NavDataFile[SVnum],toe:25);
      writeln(NavDataFile[SVnum],IODE3:25);
      writeln(NavDataFile[SVnum],Cic:25);
      writeln(NavDataFile[SVnum],OMEGAo:25);
      writeln(NavDataFile[SVnum],Cis:25);
      writeln(NavDataFile[SVnum],io:25);
```

```
111       writeln(NavDataFile[SVnum],Crc:25);
112       writeln(NavDataFile[SVnum],omega:25);
113       writeln(NavDataFile[SVnum],omegadot:25);
114       writeln(NavDataFile[SVnum],IDOT:25);
115       writeln(NavDataFile[SVnum],LastMeasNumber:10);
116       writeln(NavDataFile[SVnum])             {empty line}
117    END;
118    close(NavDataFile[SVnum]);
119 END;              { of PROCEDURE write_navdata_to_file }
120
121 (* ----- END OF THE FILES OPERATION FOR THE NAVIGATION DATA ----- *)
122
123
124
125 (* ----- THE FILES OPERATION FOR THE RAW DATA ----- *)
126
127 PROCEDURE open_files_for_magnavox (input_file,
128                                    output_file : STRING);
129 BEGIN
130    assign(f_in,input_file+'.dat');
131     reset(f_in);
132    assign(futt,output_file+'.utt');
133     rewrite(futt);
134    assign(fnav,output_file+'.nav');
135     rewrite(fnav);
136    assign(fpos,output_file+'.pos');
137     rewrite(fpos);
138    assign(fsat,output_file+'.sat');
139     rewrite(fsat);
140    assign(ferr,output_file+'.err');
141     rewrite(ferr);
142    reset_navdata_files(output_file, NavDataFiles);
143 END;
144
145
146 PROCEDURE reopen_output_files;
147 VAR
148    i : byte;
149
150 BEGIN
151    rewrite(futt);
152    rewrite(fnav);
153    rewrite(fpos);
154    rewrite(fsat);
155    rewrite(ferr);
156    FOR i:=1 TO max_sv_id DO
157    BEGIN
158      rewrite(NavDataFiles[i]);
159      close(NavDataFiles[i]);
160    END;
161 END;
162
163
164 PROCEDURE write_rawdata_to_file (    ACM  : AllChannelMeasType;
165                                  VAR futt : text);
166 VAR
167    i : byte;
168
169 BEGIN
170    write(futt,ACM.user_ms/1000:10:0,' ');
171    FOR i := 1 TO max_channels DO
172    BEGIN
173       WITH ACM.channel[i] DO
174       IF (PRN <> 0)                  {0 means: no information in this channel}
175       THEN BEGIN
176          write(futt, PRN:4,' ');
177          IF time_corrected      = TRUE
178             THEN write(futt, time_of_transmission:25:16,' ');
179          IF pr_state            = TRUE THEN write(futt, PseudoRange:15:4,' ');
180          IF carrier_phase_state = TRUE THEN write(futt, int_phase:15:8,' ');
181          IF code_phase_state    = TRUE THEN write(futt, RawCode:5,' ');
182          {write(futt, pr_receiv);}
183          write(futt,NoOfHatch:5,' ');
184       END
185       ELSE BEGIN
186          write(futt,'  0 ');
187          write(futt,'           0 ');
188          write(futt,'                    0 ');
189       END;
190    END;                {for chnl}
191    writeln(futt);
192 END;              { of PROCEDURE write_data_channels_to_file }
193
194
195 PROCEDURE close_files_for_magnavox;
196 BEGIN
197    close(f_in);
198    close(futt);
199    close(fnav);
200    close(fpos);
201    close(fsat);
202    close(ferr);
203 END;
204
205 (* ----- END OF THE FILES OPERATION FOR THE RAW DATA ----- *)
206
207
208 (* ----- THE FILES OPERATION FOR DIFFERENTIAL CORRECTION ----- *)
209
210 PROCEDURE open_files_diff_correction (input_file,
211                                       output_file : STRING);
212 BEGIN
213    open_files_for_magnavox (input_file, input_file);
214    assign(fpre, output_file+'.pre');
215     rewrite(fpre);
216 END;
217
218
219 PROCEDURE write_to_file_pre (result : ResultType;
220                              VAR fpre : text);
```

```
221 VAR
222    i : BYTE;
223
224 BEGIN
225    write(fpre,result.UncTOR:10,' ',result.EstTOR:25,' ');
226    FOR i:=1 TO 6 DO
227    WITH result.channel[i] DO
228      write(fpre,PRN:3,' ',PRE:25,' ',NoOfHatch:3,' ');
229    writeln(fpre,result.AveragePRE:25);
230 END;
231
232
233 PROCEDURE close_files_diff_correction;
234 BEGIN
235    close_files_for_magnavox;
236    close(fpre);
237 END;
238
239 (* ----- END OF THE FILES OPERATION FOR DIFFERENTIAL CORRECTION ----- *)
240
241
242 (* ----- THE FILES OPERATION FOR THE POSITION CALCULATION (gps) ----- *)
243
244 PROCEDURE open_files_rec_position (input_file,
245                                    output_file : STRING );
246 BEGIN
247    open_files_for_magnavox (input_file, input_file);
248    assign(frec,output_file+'.rec');
249     rewrite(frec);
250 END;            { of PROCEDURE open_files_sat_position }
251
252
253 PROCEDURE write_to_file_rec (    NumOfSVs : byte;
254                              VAR Rec      : ResultType);
255 BEGIN
256    write (frec,Rec.UncTOR:10);
257    write (frec,Rec.EstTOR:20:12);
258    WITH Rec DO
259       write (frec,Position[1]:14:3,' ',Position[2]:12:3,' ',
260                   Position[3]:12:3,' ',Position[4]:12:5);
261    writeln(frec)
262 END;           { of PROCEDURE write_to_file_rec }
263
264
265 PROCEDURE close_files_rec_position;
266 BEGIN
267    close_files_for_magnavox;
268    close(frec);
269 END;
270
271 (* ----- END OF THE FILES OPERATION FOR POSITION CALCULATION (gps) ----- *)
272
273 END.
274
275 (* ------------------------ END OF FILE -------------------------------- *)
```

```
(* ------------------------ FILE COMMON.PAS ---------------------------- *)

UNIT common;

(* ************************************************************************ *)
(*                                                                          *)
(*   This unit contains the functions and the procedures that used in all   *)
(*   of the programs.                                                       *)
(*                                                                          *)
(* ************************************************************************ *)

INTERFACE
{$N+}

USES
  crt,
  types;

PROCEDURE BuildUpScreen;
PROCEDURE buildup_screen_position;
PROCEDURE PrintScreenCurrentMeasNumber (k : longint);
PROCEDURE PrintScreenMeasData (acmt : AllChannelMeasType);
PROCEDURE print_screen_user_position (res_pos : ResultType);
PROCEDURE print_screen_nav_data ( VAR SVnum  : byte;
                                  IOD    : byte);
FUNCTION ReadFromKeyBoard ( NumberOfChar : byte;
                            NumberTrue   : boolean;
                            Default      : STRING) : STRING ;
PROCEDURE ResetAllChannelMeasType ( VAR acmt : AllChannelMeasType);
PROCEDURE reset_state_condition;

PROCEDURE navdata_to_arraynavdata (SVnum    : byte;
                                   NavData  : NavDataType;
                                   VAR ANDT : ArrayNavDataType);

FUNCTION GPStimeIncrement ( OldTime,
                            NewTime : float) : float;
FUNCTION MeasNumberIncrement ( NewMeasNo,
                               OldMeasNo : longint) : longint;
PROCEDURE ExtractTimeFromMeasNumberMS (    MNms : longint;
                                       VAR s    : STRING);


IMPLEMENTATION

VAR
   HatchFil,                            {Hatch filtering over n measurements}
   MeasMissing      : integer;
   StoreEMx,
   StoreEMy,
   StoreNDx,
   StoreNDy         : byte;                     {used in PrintScreenNewNavData}


PROCEDURE BuildUpScreen;
BEGIN
   CLRSCR;
   writeln;
   writeln;
   writeln(' GPS system time    : ');
   writeln;
   writeln(' Measurement Number : ');
   gotoxy(1,10);
   write('   Chan    PRN    Pseudo range    Sig/No');
END;


PROCEDURE buildup_screen_position;
BEGIN
   gotoxy(43,3);
   writeln (' USER POSITION ');
   gotoxy(44,4);
   writeln (' X : ');
   gotoxy(44,5);
   writeln (' Y : ');
   gotoxy(44,6);
   writeln (' Z : ');
   writeln;
END;


PROCEDURE PrintScreenCurrentMeasNumber (k : longint);
VAR
   s : STRING;

BEGIN
   gotoxy(24,3);
   ExtractTimeFromMeasNumberMS(k*1000, s);
   write(s,'   ');
   gotoxy(24,5);
   write(k:12,'    ');

END;


PROCEDURE PrintScreenMeasData (acmt : AllChannelMeasType);
VAR
   i : byte;

BEGIN
   window(5,11,80,25);
   FOR i:=1 TO max_channels DO
      IF (acmt.channel[i].PRN=0)
      THEN writeln('                                                ')
      ELSE writeln ( i:2,
                     acmt.channel[i].PRN:8,
                     acmt.channel[i].PseudoRange:15:2,
                     acmt.channel[i].SNR:8);
   writeln;
   window(1,1,80,25);
```

```
END;


PROCEDURE print_screen_user_position (res_pos : ResultType);
VAR
   i : byte;

BEGIN
   window (49,4,80,13);
   FOR i:=1 TO 3 DO writeln (res_pos.position[i]:12:3);
   window (1,1,80,25);
END;


PROCEDURE print_screen_nav_data ( VAR {MeasNo : longint;}
                                      SVnum  : byte;
                                      IOD    : byte);
VAR
   s : STRING;

BEGIN
   window(47,10,80,25);
   {ExtractTimeFromMeasNumberMS(IOD*1000, s);
   write(s,' ');}
   writeln({MeasNo:10,}SVnum:6,IOD:6);
   window(1,1,80,25);
END;


FUNCTION ReadFromKeyBoard (NumberOfChar : byte;
                           NumberTrue   : boolean;
                           Default      : STRING ) : STRING;
VAR
   x,
   y          : byte;
   ch         : char;
   Stop       : boolean;
   TempString : STRING;

BEGIN
   x:=wherex;
   y:=wherey;
   gotoxy(x+NumberOfChar+1,y);
   write('(',Default,')');
   gotoxy(x,y);
   Stop:=false;
   TempString:='';
   WHILE (NOT Stop) DO
   BEGIN
     ch:=ReadKey;
     IF ch=#0
     THEN ch:=ReadKey
     ELSE BEGIN
       IF (ch=#8) AND (length(TempString)<>0)
       THEN BEGIN
         TempString:=copy(TempString,1,length(TempString)-1);
         gotoxy(x,y);
         write(TempString);
         write(' ');
       END;
       IF (ch=#13)
       THEN BEGIN
         Stop := true;
         IF length(TempString)=0 THEN TempString := Default;
       END;
       IF length(TempString)<NumberOfChar
       THEN BEGIN
         IF ((NumberTrue) AND (ch IN ['0'..'9']))
            THEN TempString:=TempString+ch;
         IF ((NOT NumberTrue) AND (ch IN ['0'..'9','a'..'z','A'..'Z']))
            THEN TempString:=TempString+ch;
         gotoxy(x,y);
         write(TempString);
       END;
     END;
   END;
   ReadFromKeyBoard:=TempString;
END;                    { of PROCEDURE ReadFromKeyBoard }


PROCEDURE ResetAllChannelMeasType ( VAR acmt : AllChannelMeasType );
(* ************************************************************************ *)
(*   Deletes all information in AllChannelMeasType                          *)
(* ************************************************************************ *)
VAR
   i : byte;

BEGIN
   FOR i:=1 TO max_channels DO
      acmt.channel[i].PRN := 0;
END;                              { of PROCEDURE ResetAllChannelMeasType }


PROCEDURE reset_state_condition;
BEGIN
   time_corrected      := true;
   pr_state            := false;
   carrier_phase_state := false;
   code_phase_state    := false;
END;


PROCEDURE navdata_to_arraynavdata (SVnum    : byte;
                                   NavData  : NavDataType;
                                   VAR ANDT : ArrayNavDataType);
BEGIN
   WITH ANDT[SVnum] DO
     BEGIN
       PRN := SVnum;
       IODC     := NavData.IODC;
```

```
221         IODC_MSB := NavData.IODC_MSB;
222         toc      := NavData.toc;
223         Tgd      := NavData.Tgd;
224         af2      := NavData.af2;
225         af1      := NavData.af1;
226         af0      := NavData.af0;
227         Accuracy := NavData.Accuracy;
228         Health   := NavData.Health;
229         WeekNo   := NavData.WeekNo;
230         IODE2    := NavData.IODE2;
231         Crs      := NavData.Crs;
232         deltan   := NavData.deltan;
233         Mo       := NavData.Mo;
234         Cuc      := NavData.Cuc;
235         e        := NavData.e;
236         Cus      := NavData.Cus;
237         Asqrt    := NavData.Asqrt;
238         toe      := NavData.toe;
239         IODE3    := NavData.IODE3;
240         Cic      := NavData.Cic;
241         OMEGAo   := NavData.OMEGAo;
242         Cis      := NavData.Cis;
243         io       := NavData.io;
244         Crc      := NavData.Crc;
245         omega    := NavData.omega;
246         omegadot := NavData.omegadot;
247         IDOT     := NavData.IDOT;
248      END;
249 END;
250
251
252 FUNCTION GPStimeIncrement (OldTime,
253                            NewTime : float ) : float;
254 {*************************************************************************}
255 { Returns the time increment from NewTime to OldTime
256   GPSTime : float [ 0.604800 ]
257   Result  : float [ -302400..302400 ]                                     }
258 {*************************************************************************}
259 CONST
260    MaxGPStime : float = 604800.0;           {GPS time in range 0..MaxGPStime}
261    HalfGPStime: float = 302400.0;
262 VAR
263    MND_tmp  : float;
264
265 BEGIN
266    MND_tmp := NewTime - OldTime;
267    IF (MND_tmp >=  HalfGPStime)                                  {out of range}
268       THEN MND_tmp := MND_tmp - MaxGPStime;
269    IF (MND_tmp <= -HalfGPStime)                                  {out of range}
270       THEN MND_tmp := MND_tmp + MaxGPStime;
271    GPStimeIncrement:= MND_tmp;
272 END;           { of FUNCTION GPStimeIncrement }
273
274
275 FUNCTION MeasNumberIncrement (NewMeasNo,
276                               OldMeasNo : longint): longint;
277 {*************************************************************************}
278 { Returns the increment from OldMeasNo to NewMeasNo (both ordinal types,
279   MeasNumberType = 0..MaxMeasNumber
280   Result positive if New is newer than old
281   Returned value [-MaxMeasNumber/2 .. MaxMeasNumber/2]                    }
282 {*************************************************************************}
283 CONST
284    HalfMeasNumber: longint = trunc(MaxMeasNumber/2);
285 VAR
286    MND_tmp  : longint;
287
288 BEGIN
289    MND_tmp := NewMeasNo - OldMeasNo;
290    IF (MND_tmp >=  HalfMeasNumber)                               {out of range}
291       THEN MND_tmp := MND_tmp - (MaxMeasNumber+1);
292    IF (MND_tmp <= -HalfMeasNumber)                               {out of range}
293       THEN MND_tmp := MND_tmp + (MaxMeasNumber+1);
294    MeasNumberIncrement:= MND_tmp;
295 END;           { of FUNCTION MeasNumberIncrement }
296
297
298 PROCEDURE ExtractTimeFromMeasNumberMS (    MNms : longint;
299                                        VAR s    : STRING);
300 {*************************************************************************}
301 { INPUT  : Measurement number in ms (USER_MS)
302   OUTPUT : String containing day, and (GPS time)                          }
303 {*************************************************************************}
304 VAR
305    TempString : STRING;
306    Day,
307    Hours,
308    Minuts,
309    Seconds    : longint;
310
311 BEGIN
312    MNms:=round(MNms/1000);                                       {whole seconds}
313    Day:=trunc(MNms/(24*60*60));
314    MNms:=MNms-Day*(24*60*60);
315    Hours:=trunc(MNms/(60*60));
316    MNms:=MNms-Hours*(60*60);
317    Minuts:=trunc(MNms/60);
318    Seconds:=MNms-Minuts*60;
319    CASE Day OF
320       0 : s:='Sun';
321       1 : s:='Mon ';
322       2 : s:='Tue ';
323       3 : s:='Wed ';
324       4 : s:='Thu ';
325       5 : s:='Fri ';
326       6 : s:='Sat ';
327    END;
328    str(Hours:2,TempString);
329    s:=s+TempString+':';
330    str(Minuts,TempString);
```

```
331    IF Minuts>9 THEN s:=s+TempString+':'
332                ELSE s:=s+'0'+TempString+':';
333    str(Seconds,TempString);
334    IF Seconds>9 THEN s:=s+TempString
335                 ELSE s:=s+'0'+TempString;
336 END;             { of PROCEDURE ExtractTimeFromMeasNumberMS }
337
338
339 BEGIN
340   { global variables }
341     HatchFil     := 50;
342     MeasMissing  :=  3;
343     last_meas_no := -2;
344     ResetAllChannelMeasType ( AllChannelMeas );
345     AllChannelMeas.user_ms:= -1;
346
347   { local variabels }
348     StoreEMx  := 1;
349     StoreEMy  := 1;
350     StoreNDx  := 1;
351     StoreNDy  := 1;
352 END.
353
354 (* ------------------------ END OF FILE ---------------------------------- *)
```

```
(* ------------------------ FILE TYPES.PAS ----------------------------- *)

UNIT types;

(* ************************************************************************ *)
(*                                                                          *)
(*  Function : define the constants, types and variables for al of the      *)
(*             units and the program.                                       *)
(*                                                                          *)
(* ************************************************************************ *)

INTERFACE

TYPE
   float = double;

CONST
   light_speed   = 299792458;
   L1FREQ        = 1575420000;
   L1_TO_VEL     = light_speed/L1FREQ;
   max_channels  = 6;
   max_sv_id     = 32;
   MaxMeasNumber = 604799;              {GPS time in range 0..MaxNumberOfMeas}
   TNArraySize   = 10;                                  {size of the matrix}
   pi_SEMI       = 3.1415926535898;


TYPE
  (* ------------------------------------------------------------------- *)
  (*  CONFIGURATION TYPES FOR DATA FROM THE RECEIVER                     *)
  (* ------------------------------------------------------------------- *)
   OneChannelMeasType = RECORD
      PRN                  : 0..max_sv_id;      {Sat-number; 0 is not tracked}
      phi                  : float;
      phi_frac             : float;
      int_phase            : float;
      RawCODE              : integer;
      pr_receiv            : float;
      NoOfHatch            : byte;
      HatchCODE            : float;
      PseudoRange          : float;
      time_of_transmission : float;
      SNR                  : byte;
      Ambiguity            : char;
   END;

   AllChannelMeasType = RECORD
      user_ms : longint;
      channel : ARRAY[1..max_channels] OF OneChannelMeasType;
   END;

  (* ------------------------------------------------------------------- *)
  (*  DATA TYPES FOR THE NAVIGATION DATA                                 *)
  (* ------------------------------------------------------------------- *)
{according to ICD-GPS-200 the value of pi used to convert semi-circles to rad}
                        {pi = 3.1 415 926 535 898}

   NavDataFileType = ARRAY [1..max_sv_id] OF text;  {files of navigation data}

   NavDataPointer = ^NavDataType;
   NavDataType  =  RECORD
      PRN      : byte;
    {subframe 1, RECORD type 201}
      IODC     : byte; {0..255}                              {Issue of Data clock}
      IODC_MSB : byte;                          {contains two MSB bits of IODC}
      toc      : longint;                                   {seconds, max 604784}
      Tgd      : float;                                                   {sec}
      af2      : float;                                                 {s/s^2}
      af1      : float;                                                   {s/s}
      af0      : float;                                                   {sec}
      Accuracy : byte;
      Health   : byte;
      WeekNo   : word;          {16 bit unsigned integer}                 {week}
    {subframe 2, RECORD type 202}
      IODE2  : byte;                        {Issue of Data Ephemeris subframe 2}
      Crs    : float;     {Ampl. of the sin harm. corr. term to the orbit rad}
      deltan : float;             {Mean motion difference from computed value}
      Mo     : float;                             {Mean anomaly at reference time}
      Cuc    : float;     {Ampl. of cos. harm. corr. term to the arg. of lat.}
      e      : float;                                              {Eccentricity}
      Cus    : float;     {Ampl. of sin. harm. corr. term to the arg. of lat.}
      Asqrt  : float;                       {Square root of the semi-major axis}
      toe    : longint;                             {Reference time , Ephemeris}
    {subframe 3, RECORD type 203}
      IODE3    : byte;                      {Issue of Data Ephemeris subframe 3}
      Cic      : float;  {Ampl. of cos. harm. corr. term to the angle of incl}
      omegao   : float;    {Long. of asc. node of orbit plane at weekly epoch}
      Cis      : float; {Ampl. of sine harm. corr. term to the angle of incl.}
      io       : float;                          {Incl. angle at reference time}
      Crc      : float; {Ampl. of cosine harm. corr. term to the orbit radius}
      omega    : float;                                   {Argument of perigee}
      omegadot : float;                                {rate or right ascension}
      IDOT     : float;                              {rate of inclination angle}
   END;

   AllNavDataType = ARRAY [1..max_sv_id] OF NavDataPointer;

   ArrayNavDataType = ARRAY [1..max_sv_id] OF NavDataType;

  (* ------------------------------------------------------------------- *)
  (*  DATA TYPES FOR THE SATELLITE GEOMETRY DATA                         *)
  (* ------------------------------------------------------------------- *)
   geometry = RECORD
      sat_PRN : STRING;
      azimuth : STRING;
      elevation : STRING;
   END;

   sat_geometry = ARRAY[1..max_channels] OF geometry;
```

```
111  (* ------------------------------------------------------------------- *)
112  (*  CONFIGURATION TYPES FOR REFERENCE AND USER SYSTEM PROCEDURES       *)
113  (* ------------------------------------------------------------------- *)
114  TNvector = ARRAY[1..TNArraySize] OF Float;
115  TNmatrix = ARRAY[1..TNArraySize] OF TNvector;
116
117   ECEF_pos_type = RECORD
118      x  : float;                                          {ECEF x in meters}
119      y  : float;                                          {ECEF y in meters}
120      z  : float;                                          {ECEF z in meters}
121   END;
122
123   OneChannelResultType = RECORD
124      PRN         : byte;                                  {0 if no satellite}
125      CorTOT      : float;
126      PR          : float;
127      NoOfHatch   : integer;
128      sv_position : ECEF_pos_type;
129      PRE         : float;
130   END;
131
132   ResultType = RECORD
133      UncTOR     : longint;
134      EstTOR     : float;
135      AveragePRE : float;
136      channel    : ARRAY [1..max_channels] OF OneChannelResultType;
137      position   : TNvector;
138   END;
139
140   ClockType = RECORD
141      LastUncTOR         : longint;
142      NoOfMeas           : longint;
143      ClockOffsetSeconds : float;
144      ClockDrift         : float;
145   END;
146
147  (* ------------------------------------------------------------------- *)
148  (*  CONFIGURATION TYPES FOR THE FILTER                                 *)
149  (* ------------------------------------------------------------------- *)
150   HatchDataSVType  = RECORD
151      user_ms       : float;
152      NumberOfHatch : byte;             {0 indicates, no Hatch filtering yet}
153      LastHatchCode : float;
154      LastAmbiguity : char;
155   END;
156
157   StoredHatchDataType = ARRAY [1..max_sv_id] OF HatchDataSVType;
158
159
160 VAR
161    state             : char;          {the state of the choises of the user}
162    channels_in_input : byte;
163
164 (* ------------------------------------------------------------------- *)
165 (*   Variables for the input and output files from all of the program  *)
166 (* ------------------------------------------------------------------- *)
167    f_in,                                        {inputfile comes from the MX4200D}
168    futt,             {outputfile from the converting of the measurement data}
169    fnav,                                        {outputfile of navigation message}
170    ferr,                                           {outputfile of error messages}
171    fpos,              {outputfile of the receiver position given by MX4200D}
172    fsat,              {outputfile of the satellite position given by MX4200D}
173    fpre,
174  {outputfile of the error correction that can be used in the DGPS calculation}
175    frec              : TEXT;
176                      {outputfile from the calculation of the receiver position}
177
178 (* ------------------------------------------------------------------- *)
179 (*   Variables about the satellite                                     *)
180 (* ------------------------------------------------------------------- *)
181    SV20x             : 1..max_sv_id;                  {the satellite number}
182    num_of_sv         : integer;
183
184 (* ------------------------------------------------------------------- *)
185 (*   Variables in the Hatch Filtering                                  *)
186 (* ------------------------------------------------------------------- *)
187    HatchFil,                              {Hatch filtering over n measurements}
188    MeasMissing       : integer;
189                      {number of missing measurements for out of lock detection}
190
191 (* ------------------------------------------------------------------- *)
192 (*   Variables *)
193 (* ------------------------------------------------------------------- *)
194    line_number       : longint;             {the total line of the inputfile}
195    start_meas_no     : longint;                   {-1 if start at first line}
196    end_meas_no       : longint;                      {999999 if till the end}
197    last_meas_no      : longint; {last processed measurement number, 0..604800}
198
199    data_path,
200    input_file_name,
201    output_file_name  : STRING;
202
203 (* ------------------------------------------------------------------- *)
204 (*  The boolean definitions of the converted raw data                  *)
205 (* ------------------------------------------------------------------- *)
206    pr_state          : boolean;
207    carrier_phase_state : boolean;
208    code_phase_state    : boolean;
209    time_corrected      : boolean;
210
211 (* ------------------------------------------------------------------- *)
212    refX,
213    refY,
214    refZ  : float;
215
216 (* ------------------------------------------------------------------- *)
217    AllChannelMeas    : AllChannelMeasType;
218
219    NavDataFiles      : NavDataFileType;{Output file names for navigation data}
220
```

```
221    NavData           : NavDataType;
222    ArrayNavdata      : ArrayNavDataType;
223    AllNavData        : AllNavDataType;
224
225    result,
226    RECpos            : ResultType;
227
228    ecef_svposition   : ECEF_pos_type;
229
230    ClockError        : ClockType;
231
232    UserEst,
233    UserPos           : TNvector;
234
235    Q, SVpos          : TNmatrix;
236
237    RecNum            : STRING;
238                        {record number; contains a full line of the input file}
239    IOD : byte;
240
241
242 IMPLEMENTATION
243
244 END.
245
246 (* ------------------------ END OF FILE ---------------------------------- *)
```