GALILEO NAVIGATION MESSAGE PROCESSING AND CHANNEL SIMULATION

Diplomarbeit

durchgeführt von

NEVENA DJAJA

Institut für Breitbandkommunikation
Technische Universität Graz
Vorstand: Univ.-Prof. Dr. Gernot Kubin

in Zusammenarbeit mit

Begutachter: Ao. Univ.-Prof. Dr. Erich Leitgeb
Betreuer: Dr. Wolfgang Kogler (EADS Astrium)

Graz, im August 2010

Abstract

The Navigation System Galileo is currently going through its test and development phase, which includes two satellites that broadcast navigation signals. Those signals are acquired with test receivers worldwide and deliver specially formatted files containing symbols and measurements parameters which are used for test purposes.
Galileo system in contrast to well-known GPS system, deploys channel encoding in form of convolutional coding with Viterbi decoding algorithm. The aim of this diploma thesis is to develop a software that processes raw symbols obtained after despreading. This symbol stream is still encoded, interleaved and CRC protected. The developed software recovers the navigation message and compare the results with the hardware decoded reference messages. After its successful execution, navigation parameters can be extracted from the message and then used for calculations of satellite coordinates.

Kurzfassung

Das Navigationssystem Galileo befindet sich in Entwicklungs- und Testphase, welches zur Zeit zwei Satelliten umfasst. Diese Signale können mit Testempfängern weltweit empfangen werden um daraus generierte Symbole und Messparameter in entsprechender Form zur Verfügung zu stellen.
Im Unterschied zum bekannten GPS Navigationssystem, verwendet Galileo eine Kanalcodierung und zwar Faltungskodierung zusammen mit Viterbi Decodierung. Das Ziel dieser Diplomarbeit ist es, eine Software zu entwickeln, die aus den Rohsymbolen die Navigationsnachricht gewinnt. Die Rohsymbole werden direkt nach dem Entspreizen gespeichert und beinhalten noch die verschachtelte, codierte und CRC geschützte Navigationsnachricht. Nach erfolgreicher Ausführung der Software können die Navigationsparameter aus den Symbolen extrahiert werden und zur Bestimmung der Satellitenposition herangezogen werden.

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am _____ _____

(Unterschrift)

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

_____ _____

date (signature)

Danksagung

Diese Diplomarbeit wurde im Jahr 2009/2010 bei EADS Astrium in Ottobrunn (München) durchgeführt.

Zu Beginn möchte ich mich bei Professor Erich Leitgeb bedanken, der mir bei der Suche der Diplomarbeit sehr geholfen hat, sowie bei Herrn Rudolf Kohl der es ermöglicht hat, die Diplomarbeit bei EADS Astrium zu realisieren. Weiters möchte ich mich bei meinem Betreuer Wolfgang Kogler und meinem Chef Jan Wendel bei EADS Astrium für die Hilfe und den Einsatz während der Durchführung meiner Diplomarbeit herzlich bedanken.

An dieser Stelle auch ein herzliches Dankeschön an alle Kollegen aus der Abteilung "Navigations System Engineering" von EADS Astrium in Ottobrunn.

Graz, im August 2010 Nevena Djaja

Contents

List of Figures

List of Tables

1 Introduction

Evolution of European navigation satellite system Galileo includes a couple of stages, some of them already completed successfully. This chapter provides a short description of the current state and the following phases in development of the Galileo Navigation System together with the brief presentation of the accomplished work.

1.1 Scope of work

This work describes and implements a channel model for the navigation system Galileo. The main part of this model is channel coding, namely the Convolutional coding with Viterbi decoding, for the first time employed in such navigation satellite system. It also provides an overview of the test results with navigation data that are obtained with the use of the associated software developed within this diploma thesis. Detailed description of the software is provided within the Appendices.

First part covers the simulation of one navigation channel and investigates its performance characteristic regarding the Bit Error Rate (BER) for different values of Signal to Noise Ratio (SNR). For this purpose, the structure of one navigation message type is generated as described in [21], but with random data instead of the navigation data. These random generated bits in a specified length are used for the calculation of the Cyclic Redundancy Checksum (CRC), which is then appended to the end of them. They are all encoded employing the convolutional coding algorithm and form a sequence of symbols. They are then interleaved according to the interleaving matrix in [21]. Since at the reception every acquired navigation message contains some pattern that indicates the beginning of the actual navigation page (the smallest unit that builds a navigation message), this pattern, called the synchron pattern is randomly added within the symbol sequence. Hence the simulation can draw nearer to the reality.

In the next step these symbols are transmitted over the channel, which is modeled as an Additive White Gaussian Noise channel (AWGN). This well known mathematical model is suitable for the purpose of simulation of the navigation channel because the main source of the noise in the transmission channel is the noise produced in the receiver, which is caused by thermal movements of the electrons. This effect is perfectly described using the Gaussian probability distribution for the amplitude values of the white noise having the constant spectral density (see Appendix D).

At the reception side, the steps described above are done in a reverse way; synchron pattern is found and removed, then the pages are deinterleaved and decoded employing the Viterbi decoding algorithm, and finally the CRC check is performed. These should result in the same transmitted bit sequence as from the beginning. This sequence together with the input sequence is used for calculations of BER values and their graphical representation where the obtained value from simulation is compared with the theoretical value of BER.

The second part of the work deals with the test data received from two satellites with a Real Time Receiver (RTR) for test purposes. This RTR used for experimental purposes and testing of the future Galileo signals, is referred to as the Galileo Experimental Test Receiver (GETR). It outputs files in a specific format, which description is provided in [6]. From those files two types of data sequences are extracted and processed. These are the symbol sequence that contains the received raw symbols, and the bit sequence that contains the decoded bits. Therefore, only one part of the software that was used for the simulation, namely the reception part with Viterbi decoding is applied on the sequence with test symbols. The result of this processing is the data bit sequence which should coincide with the data bit sequence that is also contained within the same file, in form of the test pages. These are already hardware deinterleaved and decoded and hence are convenient for a comparison with the data bits that appeared as output of the decoding algorithm. The results of this comparison are analyzed and in all cases they show very good correspondence.

Finally, the navigation parameters extracted from the appropriate data sequence are exported in two specific formats defined for exporting of navigation message parameters. These formats are referred to as the Receiver Independent Exchange Format (RINEX) and the Yuma format, and they are used to store different kinds of navigation parameters. They are described in one of the following chapters (also see [26] for RINEX and [5] for Yuma format). The results are again compared with the existing files in the same format and the comparison shows the correctness of the implemented routine. Values of those parameters are further used for calculations of the positions of the two existing test satellites, at which the adequate algorithm for the computation of satellite coordinates was used [20],[21]. Chapters are organized in the following structure:

Chapter 1 includes the scope of this work and an introduction to Galileo Navigation System. It highlights the current stage of design and the structure of the system as well as the plans for the future development.

Chapter 2 provides the theoretical description of the channel coding technique that is employed in Galileo, the convolutional coding with Viterbi decoding algorithm. Software implementation of this technique is the essential part of this work.

Chapter 3 represents the simulation of one navigation channel followed by the generation of one navigation message from Galileo and transmission of this message through the channel. It is the implementation of the Chapter 2 with parameters defined as in [21]. Therefore, the implemented structure is a good representation of the future Galileo navigation signal and navigation message. After reception of this signal and decoding of the extracted symbols, the results are evaluated and the Bit Error Rate (BER) is computed.

Chapter 4 depicts the structure of navigation messages that are freely and openly accessible to public users, as well as the signals carrying those messages. These messages are then evaluated in the subsequent chapter. The encrypted messages that have limited accessibility and that are provided only to specific group of users are not considered within this work.

Chapter 5 describes briefly the test receiver used for measurements and reception of navigation signals broadcast from the two test satellites. As mentioned above, the receiver is named Galileo Experimental Test Receiver (GETR). Test files that are analyzed in this work were acquired with the GETR that was placed in Weilheim, a town in Germany in the south of Bavaria. These files are examined and processed, and the results are evaluated and discussed

within this chapter. Hence, it includes the application of the second part of the simulation software, the reception part, on the test data from GETR.

Chapter 6 comments two possible output formats for export of navigation parameters extracted from navigation messages, RINEX and Yuma. It shows some examples of generated files with navigation parameters, as well as the examples of the satellite coordinates calculated from those parameters.

Chapter 7 contains the conclusion of the work and the future aspects within this area of research.

In **Appendices A, B, C** and **D** the thorough descriptions of the implemented software versions are provided as well as the mathematical explanation of the formulas used for the channel implementation, respectively. Software used for the accomplishment of this work is MATLAB (R2009a).

1.2 Overview of Galileo system

Galileo is a Global Navigation Satellite System (GNSS) established by European Union (EU) and European Space Agency (ESA) in 2002 [4]. It is the first innovative navigation system for Europe, to be used primarily for civilian purposes. Its constellation characteristics can be taken from table 1.1:

Number of Satellites	30
Number of orbital planes	3
Nominal inclination of each orbit	56°
Nominal distance betw. satellites in one orbit	40°
Orbit altitude	23222 km
Nominal semi-major axis	29600 km
Orbital period	14h 4m 22s

Table 1.1. Galileo constellation parameters, taken from [9]

Out of 30 satellites, 27 are fully operational in Walker constellation [1], orbiting in 3 equally spaced planes in Medium Earth Orbit (MEO). The remaining 3 spare satellites (one per orbit) are non-operational, and should replace in case of failure one of the operational satellites. This is performed by shifting the spare satellite to the according position within a short amount of time (a couple of days).

Since the satellites are orbiting the Earth "faster" than Earth itself rotates around its axis, one satellite will make 1.7 orbits per sidereal day, which further means that after 10 sidereal days its ground-track will repeat. This satellite constellation, when nominally operated should provide minimum 6 visible satellites to be available for positioning to every user in any place worldwide and at any moment of time, with the appropriate receiving equipment [9].

[1] Optimal constellation for navigation satellites, here denoted as Walker Delta $56° : 27/3/1$

1.2.1 System description and development

Development of Galileo System can be divided in three phases:

- Definition and Design
- In Orbit Validation (IOV) and Development
- Full Operational Capability (FOC)

Galileo system is currently going through its test and validation phase, namely the last stage of the Definition and Design phase with two launched satellites named Galileo In-Orbit Validation Element (GIOVE), hence GIOVE A and GIOVE B. Next step will be the launch of the four satellites for the IOV and Development phase [19] .

The first one, GIOVE A, was launched in December 2005 and broadcast the future Galileo ranging signals over two channels simultaneously. Its purpose was to test and validate signal frequencies that are going to be used in the future Galileo system, to test the two on-board atomic clocks (Rubidium Atomic Frequency Standard clocks - RAFS) and to characterize the orbital parameters such as radiation environment of Medium Earth Orbit (MEO), where it was placed. Currently it is in the graveyard orbit, but still operational for short periods of time [20].

The second one launched in April 2008 and named GIOVE B, is a further improved version of GIOVE A, especially from a hardware perspective regarding the signal generation. Upgraded Navigation Signal Generator Unit (NSGU) is implemented in its payload and it can generate new type of signal modulation, using the new technique called Multiplexed Binary Offset Carrier (MBOC). It can transmit over three channels, up to two simultaneously. The main challenge is the provision of higher signal accuracy in multi-path and interference environments. GIOVE B is also placed in MEO, and it carries three atomic clocks (two RAFS and one Passive Hydrogen Maser clock - PHM); one of them, the PHM, used for the first time on a space mission, should provide the better long-term accuracy [20].

In particular, the user receiver should be tested and the center frequencies and bandwidths should be verified. Both satellites are tracked and their signals are being received using Galileo Experimental Test Receiver (GETR) - that is the Galileo/GPS compatible multi-frequency receiver [6].

The next stage is the IOV phase, as mentioned above, and the launch of four operational satellites. This will be the first satellites on the final Galileo constellation. The launches are scheduled for the end of the year 2010 and the beginning of the year 2011. This system will deliver all Galileo services with full specifications.

The last phase, FOC includes launching the remaining 26 satellites and therefore achieving the full operational constellation. Long term operations and the maintenance phase of Galileo system where it will provide complete services to the users with appropriate user equipment is planned to last about 20 years [9].

One of the goals of Galileo is on one hand to be independent of the currently existing and full operational navigation system GPS, and on the other hand to be interoperable i.e. compatible with it. Hence the signals that GETR tracks and receives are not only the Galileo/GIOVE signals, but also the GPS signals [6].

1.2.2 GIOVE System Architecture

Navigation System architecture can generally be divided into three parts, each one executing its determined function and all working coordinated together to enable the full capability of navigation services. These parts are *Ground Segment*, *Space Segment* and *User Segment*.

Since Galileo is currently in its test and validation phase, the present architecture is the one of the GIOVE system [20] (Infrastructure for Galileo is in process of development) and it is shown schematically in figure 1.1. The *User Segment* is to be introduced after the full Galileo implementation, and it can now be seen as the *Test segment*.

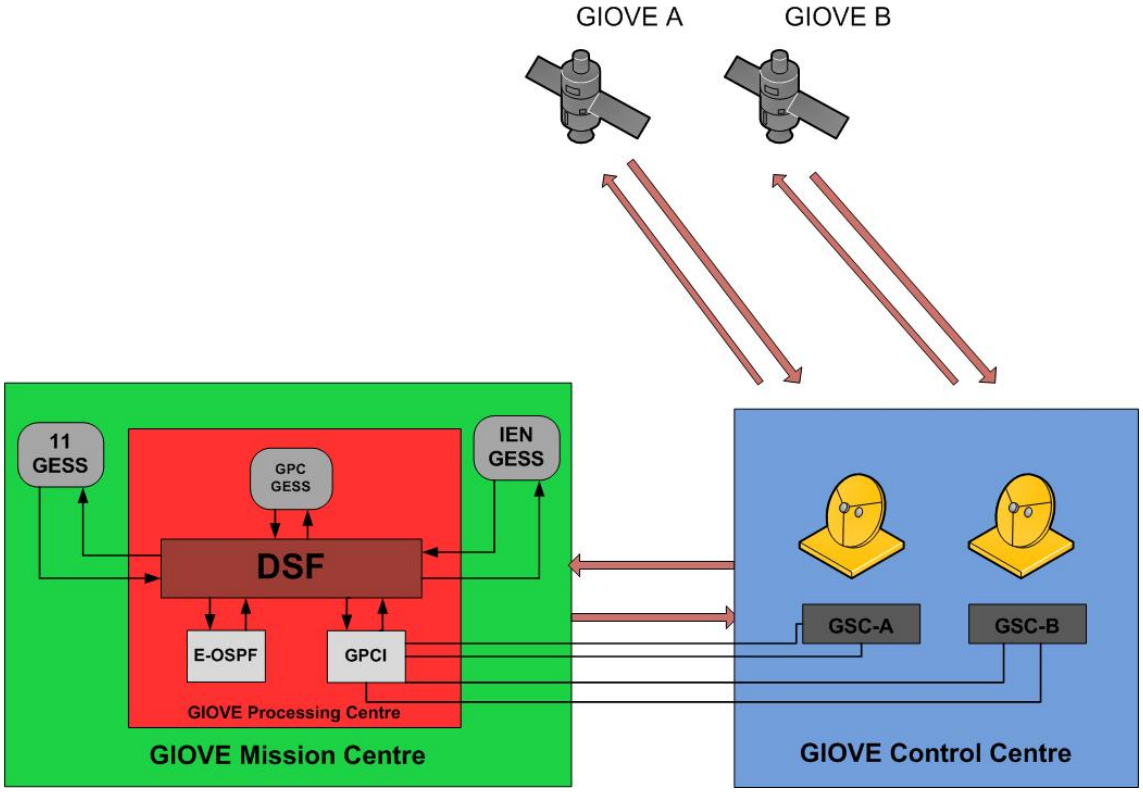

Figure 1.1. GIOVE system architecture

Ground infrastructure of the GIOVE system consists of 2 main segments, the GSC and the GMS [20],[19]:

1.GCS - Ground Control Segment
It is in charge of the control of both, the whole satellite constellation and the individual satellites. Maintenance and monitoring are the key features that are done in Telemetry, Tracking & Control (TT&C) stations network using the TT&C uplinks.

2.GMS - Ground Mission Segment
Its task is to control and monitor the main issues concerning the navigation mission, which includes computation of parameters for orbit determination and clock synchronization.

There are different facilities comprising the ground segment and they are allocated world-wide. The functions of the ground segment are accomplished with the following structure (also see the webpage in [20]):

GESS Galileo Experimental Sensor Stations - there are 13 GESS stations located all over the world and constantly acquiring signals broadcast by the satellites. They receive the signals coming from GIOVE test satellites and GPS constellation satellites. They generate raw data, in files of 15 min duration, that is converted and stored in Receiver Independent Exchange Format (RINEX). This data is delivered to GIOVE Processing Center (GPC) for further processing. From 13 stations worldwide, eleven of them are connected to the GPC through Sensor Stations Data Servers (SSDS), and the other two are directly connected to the GPC (one located in Italy and the other one inside of the GPC internal network). **GPC** GIOVE Processing Center - its main function is to configure and generate navigation data. It collects informations from GESS stations and uses them to calculate parameters that are going to be uploaded within the navigation message. Beside one GESS station, it contains three important entities, each one executing the corresponding function:

1.Data Server Facility (DSF) is responsible for data handling and archiving, monitoring and controlling the system, managing the users and executing the routine processes.

2.Experimental Orbit and Synchronization Processing Facility (E-OSPF) uses the data received from GESS stations to evaluate the orbit and clock parameters that are then employed in generation of navigation message.

3.GIOVE Payload Control Interface (GPCI) provides communication between the GPC and the two GSC centers. It collects telemetry data, orbital characteristic data, flight dynamic and attitude data etc. It transfers data to the upload station network.

GSC GIOVE Satellite Control Center - there are two control centers, GSC-A for GIOVE A, and GSC-B, for GIOVE B, and each one of them executes two functions:

a)Together with the TTC stations satellites are monitored and controlled.

b)When the navigation message is generated in GPC with the data collected from GESS stations, it is uploaded to the satellites, in order to be disseminated to the users. It collects and checks data every day periodically. This data is then uploaded over the S-band communication link.

Space segment comprises two satellites, GIOVE A and GIOVE B, as described above. GIOVE A has currently reached its end of life, and it is already shifted to another orbit (graveyard orbit), to release the place for new satellites which will be launched within the next years in order to form the full Galileo constellation. GIOVE B is still providing test signals. In the scope of this work, navigation data broadcast from both satellites, GIOVE A and GIOVE B for test purposes is being analyzed and evaluated.

1.2.3 Galileo System Architecture

When achieving its full operational capability phase, Galileo system will have all three mentioned segments, *Ground segment*, *Space Segment* and *User Segment* [18].

On the ground two segments will be active, the Ground Control Segment (GCS) and Ground Mission Segment (GMS), as described in GIOVE section. These will be controlled by Ground Control Center (GCC), which is planed to be geographically distributed in three different locations in Europe [4].

GMS will contain two separate entities, Galileo Sensor Station (GSS) network and Uplink Station (ULS) network. Without going into details, here the short review of their characteristics:

GSS There are 40 stations envisaged for Galileo, each one receiving either navigation/integrity information or just the integrity information employing two chains of receivers.

ULS There are 9 stations envisaged for Galileo, each one having 5 or 6 antennas responsible for the uplink of the mission data in C-band. GCS will include 5 TTC stations that will control the communication with satellites through uplink and downlink processes of the TTC data in S-band.

Space segment includes 30 satellites, as given above, and should be accomplished during the following view years, with the participation of several large European satellite companies.

User segment should provide services to users, depending on navigation signals. Local receiver technologies, various user applications and value added services compose the User segment.

Galileo is a civil navigation system controlled by civil institutions, hence the high accurate signal will be available to all users possessing the appropriate receiving equipment. Here it should be distinguished between the encrypted services that are provided only to specific user groups, and the open services that are freely accessible for all user groups without a service fee.

Further readings about this topic are provided in [9], [21], [20], [4] and [22].

2 Convolutional coding with Viterbi decoding technique

Galileo is the first navigation system that employs a channel coding technique for protection and correction of possible errors during the transmission, and as already stated this is the main topic of this work. For the reasons of better understanding, the theoretical description of this method is first given within this chapter, and in the next chapter its implementation in the simulation of a navigation channel.

2.1 Forward Error Correction

Channel coding is a class of signal transformations that is deployed to improve communication performances of the transmission channel regarding impairments of the channel, such as noise, interference and fading. Therefore, it is denoted as Forward Error Correction (FEC) and comprises a number of techniques applied on a communication/transmission channel to detect and correct errors that occurred during transmission of the signal over the channel. Basically all of this methods add redundancy in a specific, determined way to the data being transmitted and hence protect them. Their goal is to provide error-free (or with very low bit error rate) transmission over noisy channels. Since the procedure is performed in the receiver and since there is no need for sending request for retransmission in case of errors; it is referred to as the one-link (as its name declares, Forward) correction [16], [10].

As the two main problems in one satellite channel are the limited transmitter power and the permitted bandwidth [16], by adding redundant bits to the signal, two effects can be observed; on one hand it will reduce the required power at the receiver(expressed in terms of Signal to Noise Ratio, SNR), while on the other hand it will require a lot more bandwidth to transmit the coded signal. Therefore, some trade-off should be made. For satellite channel applications, more important is the gain of the transmitter power.

Figure 2.1 shows a simplified block scheme of one satellite navigation channel (also see [25]) and each block represents its own independent functionality.

Encryption Encoder is optional and it is used to scramble the information data, so it can not be detected by the person not knowing the encryption code. In Galileo there are two encrypted signals, with commercial and governmental encryption, however these are not examined within this work.

Channel Encoder is responsible for error detection and correction. Channel coding adds redundancy in a determined, structured way to provide the necessary protection of the data conveyed through the channel. There are a lot of types of channel coding such as Convolutional coding, Block coding, LDPC, Turbo codes etc. All of them deliver the so called message or information symbols (or just symbols - it is the term that is going to be used in

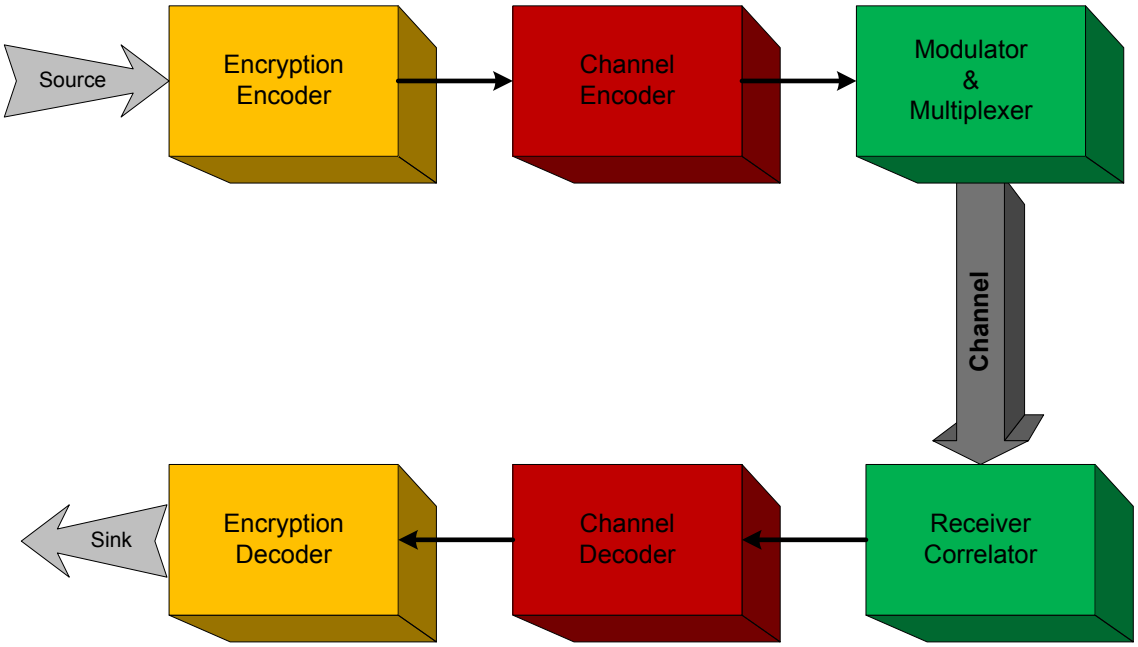

Figure 2.1. Block Scheme of Satellite Channel

this work and refers to the encoded bits), after encoding the input bit sequence. Here the convolutional encoder is used.

Modulator converts the symbol sequence into the signal form most suitable for the transmission over the channel. It can be continuous in time and bounded in some specific frequency band. In the case of satellite navigation channel, this signal form actually represents one signal component, and more signal components are going to be multiplexed to form the final composite signal, which is then broadcast over satellites (for details about the broadcast signals see [21],[13]).

Multiplexer combines the signal components obtained after modulation of symbol sequence, according to the given multiplexing scheme. Result of this operation is the signal in a certain frequency band and with the certain carrier frequency. Multiplexer together with Modulator is presented as one block.

Channel is the actual transmission medium. It can be wired, such as telephone lines, Internet cables, optic lines; or wireless such as mobile and satellite channels. In satellite navigation, channel can be seen as the "space" between the satellites and the broad network of ground stations.

Receiver Correlator correlates the received signal with the one generated with the signal generator inside the receiver and looks for the best match that corresponds to the transmitted signal. Its functionality can be represented as a more complex structure, however it does not enter into the scope of this work.

Channel Decoder corrects possible errors that might have occurred during the transmission. It decodes the symbols sequence depending on the encoding method employed at the transmission part of the channel. Its performance will depend on the characteristics of

the decoder as well as of the Signal to Noise Ratio (SNR) of the received signal. Channel decoder employed in Galileo is the Viterbi decoder.

Encryption Decoder is responsible for decryption of encrypted data. As already mentioned, it is not relevant for this work, since the encrypted signals were not processed with the developed software.

Convolutional code deals with the serial data on one or a few bits at a time. It is usually forced into some block structure, i.e. it is applied on appropriate length of the input bit sequence. There are generally three well-known ways of decoding convolutional codes; the Sequential decoding, the Threshold decoding and the Viterbi (Maximum Likelihood) decoding [10],[24] and [15].

Although a relatively new coding method, called Turbo coding took place of the convolutional coding for its better performance and its capability of achieving very low Bit Error Rates (BER) for small values of Signal to Noise Ratio (SNR) with information rate very close to the Shannon limit [1]; the convolutional coding together with the Viterbi decoding is still widely used in satellite applications.

Encoding is done on-board the satellite, where the signal is generated [21] and it is not so complex compared to the decoding which is done in the receiver equipment, since it requires a lot of memory registers.

In principle the term *channel coding* usually refers to FEC techniques together with the bit interleaving, which is not a coding technique, but only the permutation of bits in some specific manner. It is done after the encoding procedure, for better protection of data since the convolutional code is not capable of correcting very large burst errors [16].

2.2 Convolutional Coding

Convolutional coding methods date from the year 1955 [24] and appeared as alternative to Block codes widely used during that time. The first application of convolutional codes together with block codes (one as so called “inner” code and the other one as “outer” code) was in deep space applications, since it was efficient method to achieve very low error probabilities.

In satellite communications and now in navigation, convolutional codes together with Viterbi decoding are used due to their good performance and their advantageous characteristic that the decoding is accomplished after some deterministic period of time. From the hardware perspective, convolutional codes are quite easy to implement, since they only contain memory registers and some logic circuits, usually *xor* (see figure 2.2).

Convolutional coding is a linear coding method and can be *systematic* or *non-systematic*. The first one refers to a code where the unencoded input sequence is a part of the output sequence (it does not enter the *xor* logic, it just appears at one of the outputs) and it is almost always a *recursive code*. Recursive means that one of the outputs is added to the input recursively. The second coding method, *non-systematic* is the opposite one, and usually *non-recursive* [25].

[1] Shannon limit is a theoretical limit that describes the maximum possible channel capacity i.e. information rate that one given transmission channel can achieve

2.2.1 Encoder Description

In general, output of the convolutional encoder can be found as the convolution of input bits and the states of the linear shift register of the encoder [16],[10],[11],[7]. A simple convolutional encoder is given in figure 2.2.

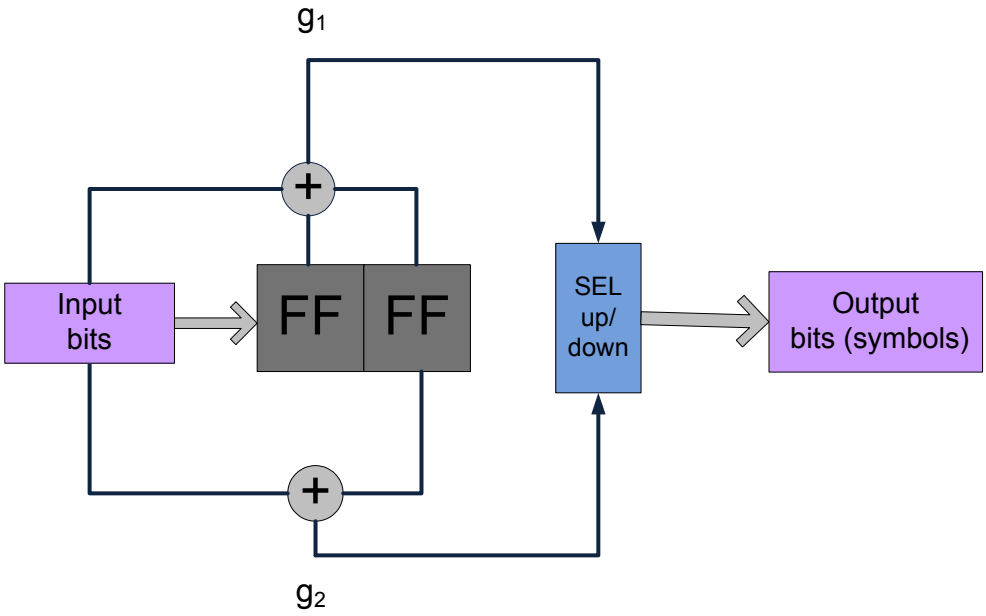

Figure 2.2. Convolutional encoder

The shift register comprises two latches (register cells, or flip-flops) and represents the states of the encoder. In every cycle of time (corresponding one clock edge) one bit enters one register cell at appropriate input and one bit exits at the output of the same register cell and enters the next one. Hence with every cycle bits are shifted through the register. The encoding is performed with combinatorial logic that is established by the use of generator polynomials (here two, g_1 and g_2), implemented with *xor* gates. Output of each gate is a combination of input bits that are currently in the register and the input bit (in this example as well as usual just one input bit at each time is shifted into encoder) that are entering the first register cell [11]. Parameters that describe the encoder are:

- **Generator polynomials** - implemented with *xor* logic and shift register. The number of the non-zero polynomials represents the number of encoded bits in each time cycle, and the highest degree in polynomial represents the length of the shift register. The notation for generator polynomials is typically octal.
- **Constraint length** - it is the length of the shift register, which is used to store the states of the encoder plus one input bit that is entering the first register cell in each time cycle. If r is representing the length of register, then $K = r + 1$ is the constraint length. The output i.e. the encoded sequence will depend on this number of bits.
- **Code Rate** - it is defined as a number of input bits to the encoder divided with the number of symbols (encoded bits) that outputs the encoder each time it produces them, hence in each time cycle. It is denoted as $CR = k/n$ and represents the measure of

efficiency of the code. Since the code adds redundancy, the CR will always be smaller than one; since $k < n$ implies that $CR < 1$.

In the example above, these parameters have the following values:

$$G_1 = 111, G_2 = 101, K = 3, CR = 1/2 \tag{2.1}$$

Where G_1 and G_2 are the first and the second generator polynomial, respectively and they are written in binary form, because of simplicity. K is the constraint length and CR the Code Rate of the code.

The encoding process can be shown by employing one of the three (or more) possible diagrams such as Finite State machine, Trellis Diagram and/or Tree structure [10],[12].

2.2.2 Finite state diagram

In this representation, shown in figure 2.3, the different states of encoder are written in circles (nodes) and transition from one state to another as lines connecting the circles. This is a static representation, that gives no information about the time. One state shows the content of each shift register in corresponding period of time (one time cycle). The transition happens every time a new bit enters the encoder.

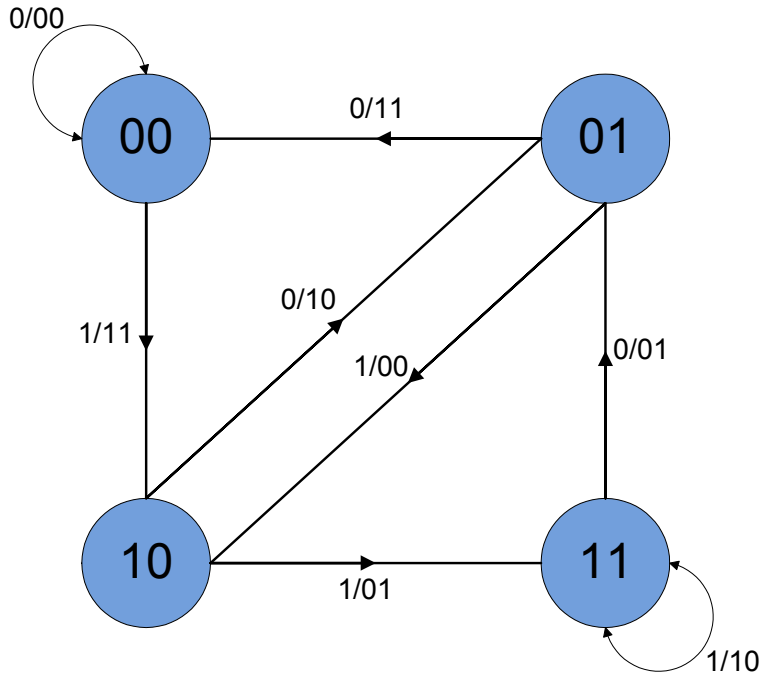

Figure 2.3. State Diagram

In this example, the states are $00, 01, 10, 11$ and the possible transitions are as noted in the diagram. Possible outputs for each transition together with the input bit entering the encoder are written on the lines connecting the states (input bit/output bits). Since there are only two different bits at the input, zero or one, maximum two transition lines will exit and enter each

state. Note that not every state transition is allowed, and not every possible output will result from it. Only the allowed outputs are the correct ones.

2.2.3 Tree diagram

Information about the time flow is present in the tree diagram, as in the figure 2.4, together with all possible states and outputs of the encoder. The encoded output bits are written on the branches of the tree, where the upper blue branch corresponds to the input bit zero and the lower red branch to the input bit one. The states are written on the nodes, using the letters a, b, c, d, where $a = 00$, $b = 10$, $c = 01$, $d = 11$. With every time instant, the tree is expanded and new paths are added. Which path is going to be followed depends on the input bit that enters the encoder.

After some depth or after the tree reaches some determined expansion of branches, the states seem to repeat, as seen in the vertical lines containing the nodes. For instance, with four states in the example, these will start to repeat after the fourth "group" of branches is achieved, i.e. starting with the fourth time instant, t_3.

It is unfavorable to work with tree diagram, especially when working with big codes, since the tree will expand to infinity. Better solution in this case would be the Trellis diagram representation.

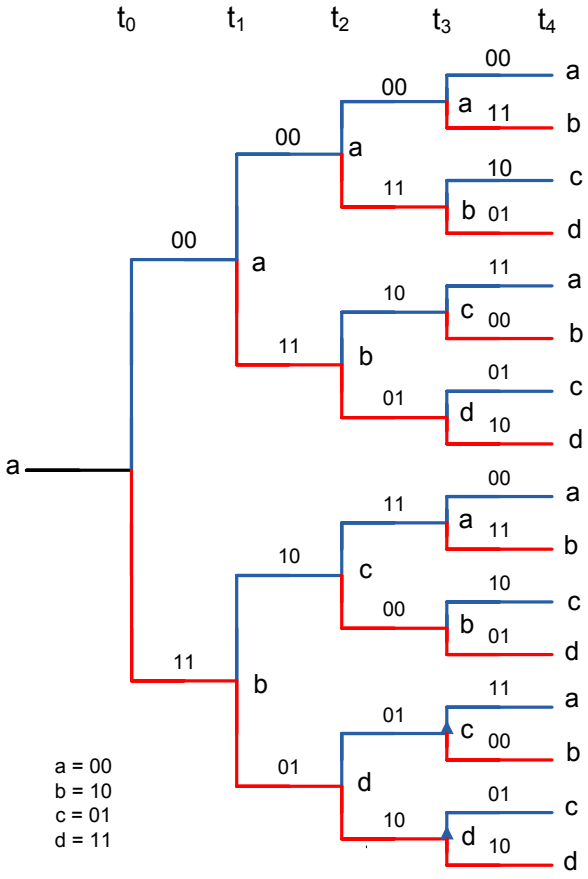

Figure 2.4. Tree Diagram

2.2.4 Trellis diagram

Trellis diagram also contains the time sequence of the encoding process, and also together with all possible states, transitions, and output bits. From each state or node, there are only two lines that emerge from that node, determined by one of the input bits, one or zero. For the same reasons, only two branches are merged in the same node. At the beginning of trellis (see figure 2.5) not all state transitions are reached. For every new input bit, one new transition is accomplished until the trellis reaches its full structure, here at time instant t_4. From that point it repeats horizontally.

Usually the trellis is terminated in *all zero* state, the same as it started. That is done by passing the zero bit sequence in the length equivalent to the length of the shift register (here 2) through the encoder. It is called "flushing" the encoder [11].

As new input bit arrives, one can make a unique path through the trellis, just going from one node to another, following the lines with corresponding input and output bits.

This is the most convenient representation of the convolutional coding, and it will be used in the description of the encoding as well as of the decoding process. It is also used for the software implementation of the Viterbi algorithm.

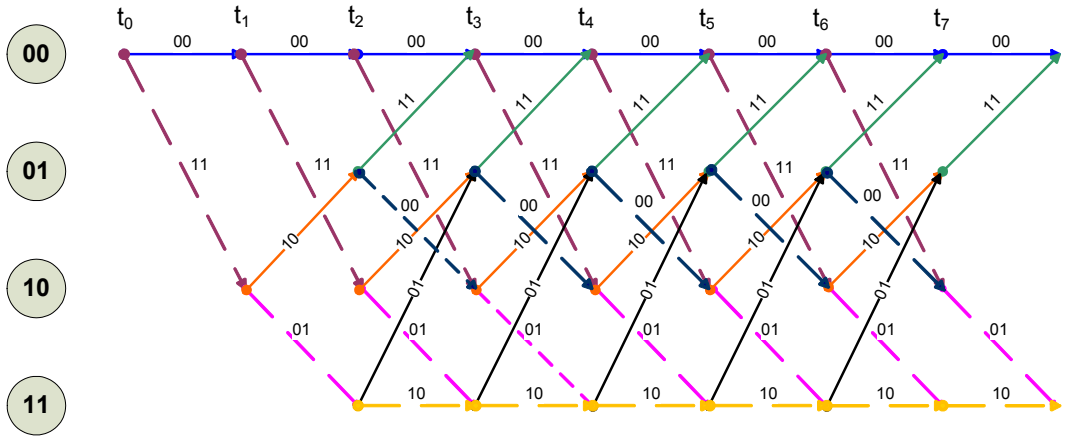

Figure 2.5. Trellis diagram

2.2.5 Encoding with Trellis

Encoding process is shown using the Trellis diagram and the example in figure 2.2 (also see [10],[16],[25],[11],[7],[12]). At each time instant t_j, one state transition is made, which means that one input bit entered the encoder, and two output bits exited it. Hence, each transition means that new bits have arrived and they are to be placed into the trellis. As the nodes refer to possible states, here four, they can be seen either in the present moment or in the moment of past (i.e. as current and/or previous state).

The trellis diagram is built up starting from the time instant $t = 0$ and state 00. From that state, there are two possibilities i.e. two transitions that can be made for two different input bits. Dashed lines stand for the input bit equal to one and solid lines for the input bit equal to zero. These branches of the trellis represent the transitions and the numbers written on the

branches are the encoded output bits (referred to as symbols or codewords) corresponding to those state transitions. The initial state is 00, hence two right-most shift register cells are filled with zeros. In the time instant from $t = 0$ to $t = 1$, there will be only 2 branches, one going to state 00 and the other one to state 10. Continuing from these two states, in the next time cycle from $t = 1$ to $t = 2$, each of them will also have 2 transitions, hence four branches will be formed. The same procedure is repeated for as many cycles as the input bits are entering the encoder, each cycle resulting in possible transitions and outputs of the codewords, written on each branch. After the depth 3 is reached (at the third time instant) the trellis structure is filled with all allowed state transitions and continues repeating periodically. In general, the constraint length K (here equal to 3)determines the trellis depth. State transitions, output pair of bits and input are shown in the table 2.1.

Input	Current State	Next State	Output
0	0 0	0 0	00
0	0 1	0 0	11
0	1 0	0 1	10
0	1 1	0 1	01
1	0 0	1 0	11
1	0 1	1 0	00
1	1 0	1 1	01
1	1 1	1 1	10

Table 2.1. State transition table with input and output entries

Now, having the input sequence $in = 01011100101000$ the path can be made through the trellis as shown in figure 2.6, where every pair of output bits on each branch represents the encoded bits and it is referred to as the branch codeword. Length of the input sequence is 14 bits, where the last two bits represent the so called *flushing bits*. Their function is to "clean" the encoder so it can finish in *all zero* state. The corresponding output sequence of encoded symbols is equal to $out = 00\,11\,10\,00\,01\,10\,01\,11\,11\,10\,00\,10\,11\,00$ and can be read directly from the encoder trellis.

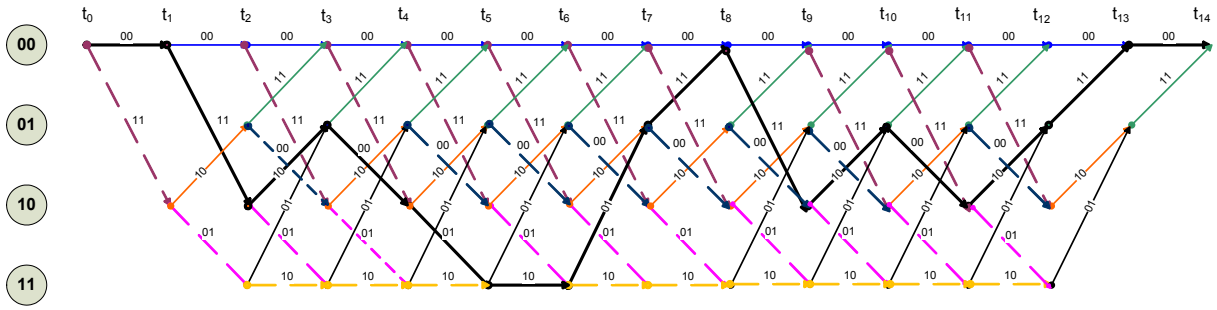

Figure 2.6. Encoder Trellis

2.3 Decoding with Viterbi-algorithm

With the full encoder trellis diagram, the decoding algorithm can be implemented. Viterbi decoder examines the entire received sequence of a given length. In this process, there are some important concepts that need to be introduced [16],[25],[11]:

HAMMING DISTANCE is a distance between two symbols that displays the number of bits at which these symbols differ.

BRANCH METRIC is the Hamming distance between every pair of the received sequence symbols and the branch codeword corresponding to that particular branch in the appropriate instant of time, extracted from the encoder trellis in figure 2.6.

PATH METRIC is a sum of metrics of all branches in the path through trellis.

The branch codewords in the trellis are nothing but the code symbols that would be expected to come as the output of the encoder for each state transition. Important issue here is that they are known to both, to encoder as well as to decoder.

Viterbi decoder relies on Maximum Likelihood concept, which is aimed at minimizing the error probability of the decoding process [16],[10],[11],[7]. Decoder does this by choosing the path in the trellis with a minimum branch metric. First it calculates the Hamming distance i.e. the branch metric between the received pair of bits and possible codewords. Then from the two paths that merge in the same state, it chooses the path with the smallest metric. Hence from eight possibilities four of them will proceed to the next step (since the choice is made four times between the two possible metrics). In the next step, this metric will be added to the previously calculated branch metrics, again eight Hamming distances and the minimum is chosen again. This leads to another important concept, namely:

ACCUMULATED PATH METRIC: It is the minimum sum of each two possible branch metrics for every time instant along the path until that time.

For example, in time instant t_3, from two branches emerging in state 00, one with the smaller metric will be chosen. Then again, in the same time instant from two branches emerging in the state 01, one with a smaller metric will enter the path metric and so on. With these accumulated smallest metric values, a path metric can be created.

This path is called the "survivor path" since it contains only the metrics that "survived" in the trellis and will be used in the decoding process. Actually this path and the knowledge about the previous state that led to this path (the one for which a minimum decision was made) are the essential part of the Viterbi algorithm. With them, one can trace back through the trellis diagram in order to find out which state transition is made at which time. Hence by knowing the received bit sequence and the state transition table 2.1, the transmitted sequence can be revealed [11].

These separate steps can easily be shown with the help of the block diagram in figure 2.7:

The logic included above is called the **ADD-COMPARE-SELECT (ACS)** procedure [16]. It is repeated for every encoder state, or equally, for every time instant.

If the input sequence from the example above enters the encoder, it will result in the following output sequence:

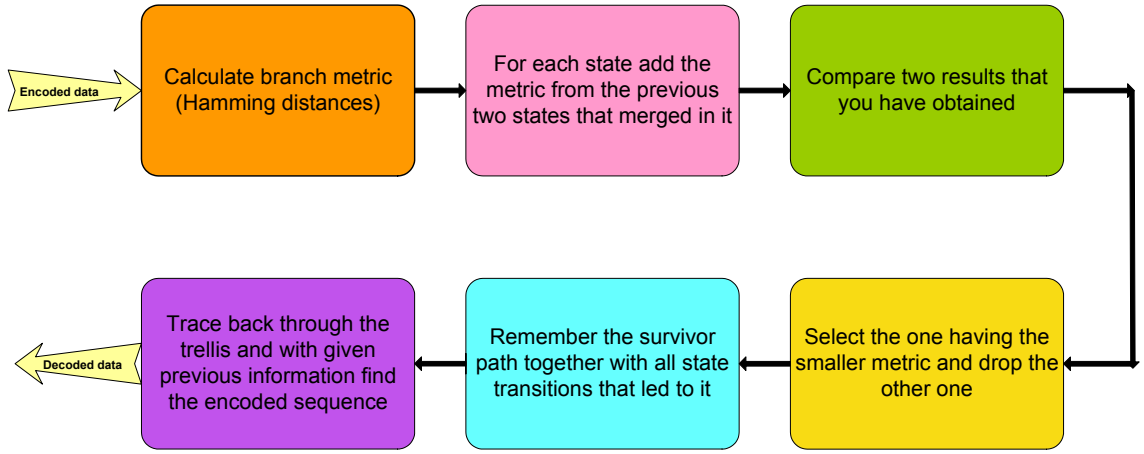

Figure 2.7. Viterbi Decoding Flow

$$in = 01011100101000,\ out = 00\,11\,10\,00\,01\,10\,01\,11\,11\,10\,00\,10\,11\,00 \tag{2.2}$$

Let assume that this sequence is affected with errors and the received sequence is:

$$rx = 00\,11\,11\,00\,01\,10\,01\,11\,11\,10\,00\,00\,11\,00 \tag{2.3}$$

First step in the decoding process will be to look at the encoder trellis diagram and to make the corresponding decoder trellis diagram as in figure 2.8.

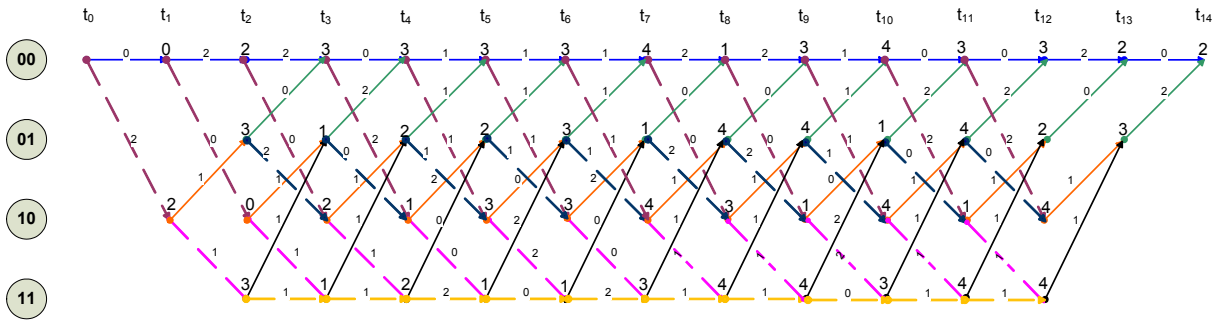

Figure 2.8. Decoder Trellis

In this diagram, the metric calculations are given for each branch, and for each state node the accumulated path metric for that node is calculated (using the ACS and the state transition table).

On the decoder trellis, one can see the metric calculations for each of the branches, as well as the accumulated path metric for each of the nodes. These accumulated metric values are used to form the table 2.2 of accumulated metric for every time instant.

As mentioned above, at the end of the input sequence $K-1$ "flushing" bits are added in order to clear the encoder and to put more coding power to the last input bit. Since every input bit affects the following $K = 3$ output pair of bits, the last one should do it too. Hence

for a given sequence of 14 input bits including the two flushing zero bits, which passes the encoder in 14 instants of time, there will be $14 * 2 = 28$ output bits (considering the code rate of $1/2$).

Table of accumulated path metric 2.2 contains minimum Hamming distances calculated for every time instant and every state.

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
State 00	0	0	2	3	3	3	3	4	1	3	4	3	3	2	2
State 01			3	1	2	2	3	1	4	4	1	4	2	3	4
State 10		2	0	2	1	3	3	4	3	1	4	1	4	3	3
State 11			3	1	2	1	1	3	4	4	3	4	2	3	4

Table 2.2. Accumulated metric table

These states are called the *survivors* and are used to determine the unencoded sequence which was transmitted. Starting from the time instant $t = 14$ the path with the minimum accumulated path metric is tracked back until the beginning of the trellis, as shown in figure 2.9.

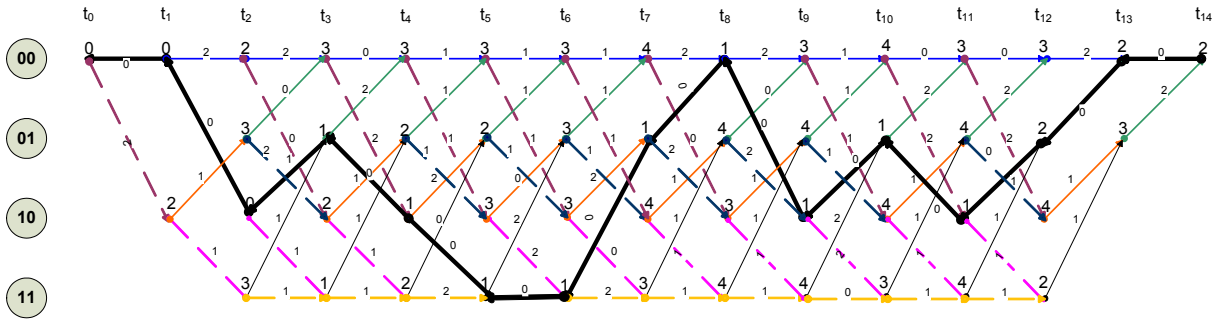

Figure 2.9. Decoder Trellis with Survivor Path

The states selected when tracing back through the survivor are listed in table 2.3. Each transition determines one input bit, starting from the beginning of the table. For example, the first bit will result from the transition $0 \rightarrow 0$, the second from $0 \rightarrow 2$ and the third one from $2 \rightarrow 1$ and so on.

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
States	0	0	2	1	2	3	3	1	0	2	1	2	1	0	0

Table 2.3. Selected states when tracing back through survivor

Now look at the state transition table from the beginning (table 2.1) and find the unencoded input bits corresponding to every state transition. The decoded bit sequence is given in table 2.4 and as it can be seen it is the same as the input sequence before it was encoded. Hence the decoder succeeded in correcting the induced errors and retrieving the original sequence of bits.

t	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
Bits	0	1	0	1	1	1	0	0	1	0	1	0	0	0

Table 2.4. Original transmitted message

Viterbi decoder is able to correct maximum K number of errors, where K is the constraint length. This example introduces the type of decoding called hard-decision decoding. It is based on hard-decision quantization that employs two quantization levels at the demodulator, one and zero, which are then fed to the decoder, and on Hamming distance computation. The other implementation, called soft-decision decoding employs a quantized value greater than two levels at the demodulator (usually eight level quantization). It feeds the decoder with more information compared to the hard-decision decoding. Instead of Hamming distance metric, the Euclidean distance is used [25],[10].

3 Implementation of channel simulation

Every transmission channel is modeled with some mathematical model that more or less demonstrates its real characteristics. In space/satellite communications as well as in satellite navigation, the essential problem to deal with is a decrement of power since the signal is passing a long way down to the Earth. There are various sources of error that can affect the signal on its way trough different transmission mediums, such as: ionospheric and tropospheric effects, interference, multipath etc [14],[22] and there are various correction models that handle them. However, the most pronounced one is the thermal noise in the receiver equipment. It is modeled with a well known Additive White Gaussian Noise (AWGN) channel model.

This chapter will demonstrate the simulation of a navigation channel which includes data transmission and reception. It implements the convolutional coding with Viterbi decoding, then Cycle Redundancy Check (CRC) and interleaving techniques as described in [21]. The obtained results are discussed in the separate section.

3.1 Simulation environment

The channel that is simulated is shown in figure 3.1 where each block represents one separate functionality. Some of the steps need to be explained carefully, for the other ones (convolutional encoder and Viterbi decoder) see Chapter 2.

3.1.1 Transmission segment

Input data are random generated bits with format of Free Accessible Navigation message, F/NAV according to [21]. Transmission segment that can be seen on the block diagram in figure 3.1 shows the steps in generation of the smallest unit of navigation message, namely the page. Every page is generated in the same way and all generated pages are concatenated to form one symbol sequence. This sequence is transmitted over an AWGN channel and in the receiver section those pages are extracted and compared with the transmitted ones, in order to prove the correctness of the transmission chain. This is done for the purpose of testing the functionality of the program, i.e. of every block that is going to be used later, when dealing with navigation data from test satellites.

Now the blocks from the transmission chain in block diagram in figure 3.1 are described in detail:

CRC stands for Cyclic Redundancy Check, which represent a Cycle Code, also called CRC code. It is a coding technique used for error detection, specially suited for burst transmission and often used in satellite communication and navigation. Like other error detection and correction techniques, it adds redundancy to input information data that is to be transmitted. The CRC algorithm generate the redundant bits by division of data bits with a specially

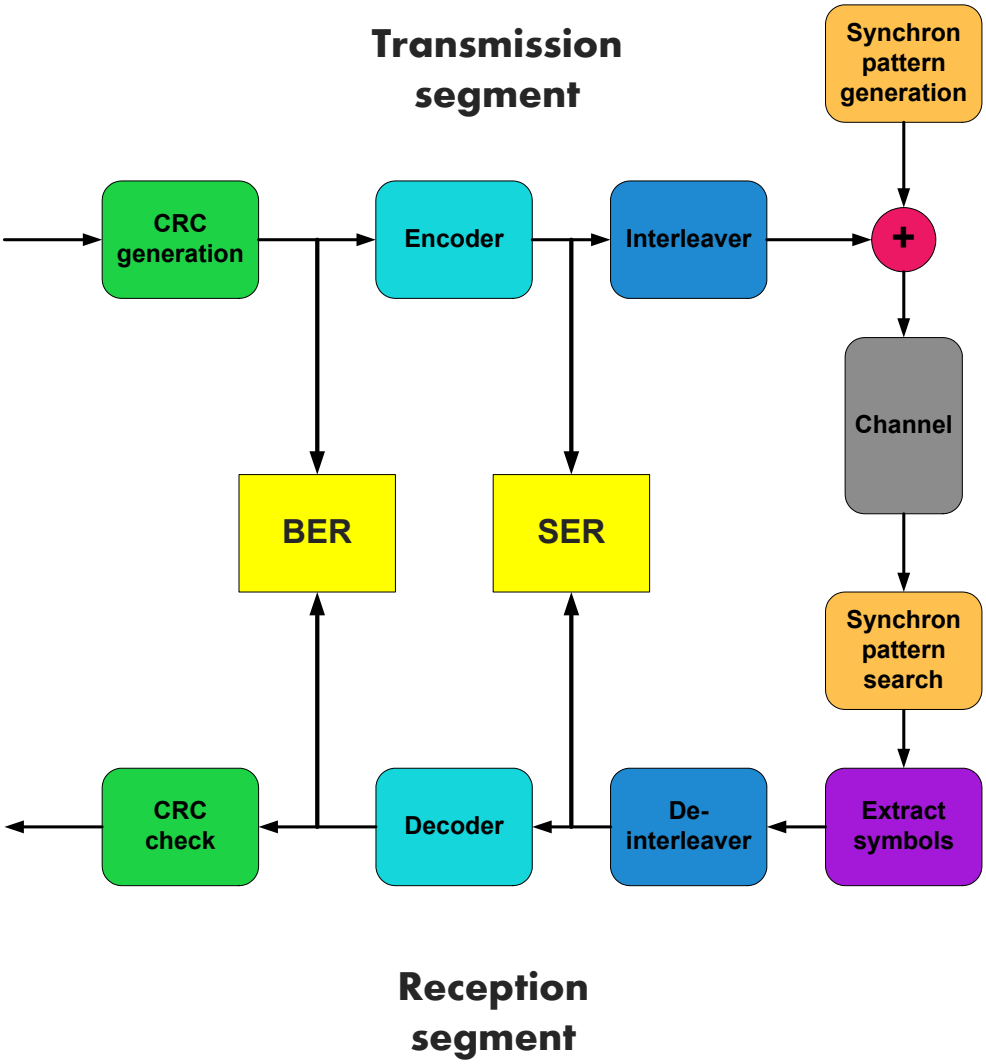

Figure 3.1. Block scheme of channel simulation

designed generator polynomial, where not the quotient but the reminder is taken into account [1]. This reminder is then appended to the input data stream that will be transmitted, and the same algorithm is employed on the received data, after the decoding process.

The result of this operation determines whether the decoding was successful (data correct and can be used) or not (data false and it is to be rejected). This information cannot be gained directly from the decoding process. Galileo system employs the following algorithm for generation of the CRC code according to [21],[1].

CRC checksum computation: Generator polynomial $G(x)$ used for the checksum computations in Galileo should be generated from the primitive and irreducible polynomial $P(x)$ as following:

$$G(X) = (1+X) \cdot P(x) \tag{3.1}$$

where:

$$P(x) = X^{23} + X^{17} + X^{13} + X^{12} + X^{11} + X^9 + X^8 + X^7 + X^5 + X^3 + 1 \quad (3.2)$$

After this binary computation, the generator polynomial is obtained as:

$$G(x) = X^{24} + X^{23} + X^{18} + X^{17} + X^{14} + X^{11} + X^{10} + X^7 + X^6 + X^5 + X^4 + X^3 + X + 1 \quad (3.3)$$

Having $m(X)$ as the sequence of input bits, CRC parity bits are computed as the reminder of the division of $m(X) \cdot X^{24}$ by generator polynomial $G(X)$. Since they are binary numbers, this operation is performed using *xor* computation. The CRC checksum will have the length of 24 bits. For better understanding, the division is shown on one small example:

Example of checksum computation: Let the generator polynomial be in a binary form 101 and the input bits used for computation 1001. Since the polynomial has a degree of $r = 3$ then the $r - 1 = 2$ zeros are appended to input bits, and the input sequence 100100 is obtained. These are then divided with the polynomial using the *xor* operation (binary algebraic division).

```
100100
101
---
001100
  101
---
  0110
   101
---
   011
```

Figure 3.2. CRC computation

In each step the following operation is made: if the input bit above the first divisor bit (MSB of the divisor) is zero, the divisor is shifted one bit to the right; if it is one, the *xor* operation is performed. This is executed until the LSB of the divisor reaches the last input bit. The result of the computation is the actual remainder of polynomial division. In this example, it is equal to 011 and this is the CRC checksum. It is appended to the original input bit sequence before transmission through the channel. In our example the sequence to be transmitted after appending the CRC checksum is 1001011. When received, the same computation is made and the received and computed CRC bits are compared. If they match exactly, the data is received correctly. If not, the received sequence is not equal to the transmitted one, and therefore not correct; it contains some errors.

In the current GIOVE test system, the CRC computation is done in the same manner as envisaged for Galileo, with a generator polynomial having the form of:

$$G(x) = X^{12} + X^{11} + X^7 + X^5 + X^3 + X^2 + X + 1 \quad (3.4)$$

and with CRC checksum 12 bits long [20].

After appending the CRC checksum to the randomly generated input bits used for the simulation, one more step has to be done, before data can enter the encoder; and that are the

six zero bits, called Tail bits, that also need to be appended to input bits after the checksum field. Their role is to "flush" the encoder as explained in Chapter 2. Since the length of the shift register of the convolutional encoder (see figure 3.1) is six, the number of the zero bits appended has to be six as well.

Encoder: Encoder is a typical convolutional encoder with characteristics as listed in table 3.1.

Code rate	1/2
Constraint length	7
Generator polynomials	$G_1 = 171, G_2 = 133(\text{octal})$
Encoder output	$G_1 G_2$

Table 3.1. Encoder parameters

These parameters have the same value in GIOVE as in Galileo, only the second output bit of encoder, namely the G_2 should be inverted when working with the future Galileo configuration [21].

The encoder used in this simulation is shown in figure 3.3:

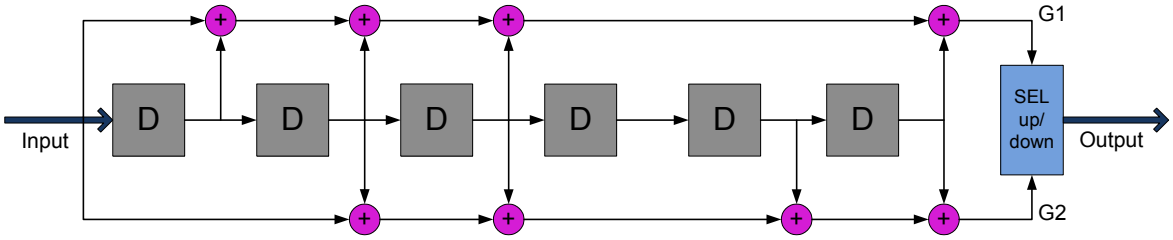

Figure 3.3. Convolutional coding scheme

Since the Code Rate is equal to $1/2$, it means that each input bit will be encoded to produce the two output bits, hence the sequence of symbols [1] that exits the encoder will have twice the length of the sequence of bits that enters it. In Galileo, as already mentioned, the smallest unit that contains navigation data is called *page*. It has a specific length depending on the navigation message type. In this simulation, the F/NAV message type is used (see Chapter 4 for details) with 244 bits entering the encoder and 488 encoded bits (symbols) obtained afterwards. This symbols, together with the synchron pattern appended at the beginning comprise one page [21].

Interleaver: For a purpose of better correction of burst errors that can occur in the channel and for improvement of the error correction performance, the interleaving is employed on the appropriate part of the sequence of symbols (on one page without the pattern), here 488 bits.

It is done with the Interleaver Block Matrix, whose dimensions will differ for each navigation message type. According to [21], the matrix dimensions for F/NAV will be 61×8, with

[1] In the following text, the encoded bits are referred to as symbols and the bits before encoding and after decoding simply as data bits

61 column bits and 8 row bits. The symbols are written into the matrix column by column and read out of the matrix row by row.

Synchron pattern generation: The essential part for synchronizing the beginning of navigation message, i.e. to determine the part containing navigation data is the synchron pattern. It indicates the beginning of the navigation page. Finding the correct position of the pattern determines also the beginning of the symbols of navigation message needed for further processing. These symbols will be decoded and deinterleaved in the reception segment. In the case of F/NAV, the pattern is 12 bits long, and together with encoded symbols one page of a total length of 500 bits is obtained.

In order to achieve a more realistic simulation environment for one navigation data transmission channel, the simulation in the case of errors is performed also with the pattern that was included in the page and modified. It provides a good test environment for different situations that can occur during transmission:

- **Case 1**: Synchron pattern is affected with errors, hence some of the bits are alternated and the pattern does not have a form as the original one.
- **Case 2**: Synchron pattern is completely missing at the beginning of one or more pages.
- **Case 3**: Synchron pattern appears in its original form within the navigation page one or more times.

All this cases are analyzed with one separate stand-alone test program. In order to analyze possible worst cases scenarios mentioned above, the pattern is added randomly at the beginning, as well as in the actual symbol sequence. In some cases instead of the pattern, a sequence of zeros is appended at the beginning of the page. Important thing to have in mind is that this operation is performed on the symbols, i.e. after the data is being encoded. Therefore, the synchron pattern does not enter the encoder!

The description of the synchron pattern search block from the block diagram 3.1 is given within the section 3.1.3. It is implemented in a final software version after the tests with worst cases are done and verified.

Before transmitting the symbol sequence over the channel, it is shifted circularly for a random number of bits, hence the first synchron pattern does not appear at the beginning, but after some period of time (some bit length).

3.1.2 Simulation channel

Channel is implemented as the Additive White Gaussian Noise (AWGN) channel model. In first step the BPSK [2] modulation is employed, where the symbols are mapped to signals with two different (antipodal) values. The symbols (0,1) are mapped to the signals with values (-1,1). Then the Gaussian white noise is added. The variance of such noise is defined according to the equation 3.5 (also see [25],[10]).

[2] BPSK refers to Binary Phase Shift Keying; it is a type of modulation where the carrier is switched between two phases by the modulating data signal

$$\sigma = \sqrt{\frac{1}{2 \cdot 10^{\frac{E_s/N_0}{10}}}} \tag{3.5}$$

with E_s/N_0 as Energy per Symbol to Noise Power spectral density ratio, defined as [25]:

$$E_s/N_0 = E_b/N_0 + 10\log CR \ [dB] \tag{3.6}$$

where E_b/N_0 is the Energy per Bit to Noise Power spectral density ratio and CR is the Code Rate of the code as defined in Chapter 2. For more details about this topic see Appendix D.

After adding noise, the signal has to pass the quantizer, which makes either hard or soft decision. Hard decision corresponds to one-bit quantization, and soft decision is implemented here as three-bit quantization [16],[10].

The steps described in the following section are done at the receiver side i.e. they simulate the reception part of the transmission channel and therefore they are executed reversely to the transmitter side.

3.1.3 Reception segment

Synchron pattern search: This is an important step at the receiver part of the channel, since the synchron pattern is intentionally added at the beginning of each block of symbols that has to be decoded and it is repeated periodically after a specific length; in this case it is the length of one page. The pattern has to be removed before symbols enter the decoder. It does not contain any useful information about navigation parameters, it acts as an indicator for a start of the page, i.e. of the part of the symbol sequence that contains navigation data.

Algorithm for searching the synchron pattern is developed with appliance of correlation function on the received symbol sequence and on the synchron pattern. Correlation function is useful to find the match of one function with another. Since the received symbol sequence contains the pattern multiple times, the result of the correlation function applied on the symbols and the pattern should represent the indices of the positions of the synchron patterns within the sequence. Recall that the symbols are affected with noise and that the pattern is added randomly at the transmission part of the channel. Therefore some additional check function should be provided in order to find the exact indices which are placed in a distance of one page length to each other. This examination is again done with the correlation function. One example is provided for this procedure, which is implemented according to the following steps:

1. Correlate one block length of the received symbol sequence with the synchron pattern for that specific navigation message type. In the example shown in figure 3.4, one block with 10 pages is considered (hence $10 \cdot 500 = 5000$ bits length), and the result of correlation is presented. The highest peaks and their distribution can be recognized easily, there are 10 of them, but there is still a lot of interference from the lower peaks. In order to be sure about the right decision, the second correlation is performed.

2. Correlate the result of the first correlation with one sequence of the same length, namely with vector of the ideally placed synchron patterns. This vector is shown in figure 3.5, where the height of the peaks is equal to the length of the synchron pattern, and they are placed in equal distances (here equal to 500) as the pattern should be placed in the block sequence, starting from the beginning. The result of this correlation can be seen in figure 3.6 - now the highest peaks are more obvious, and the other values are drawn back into the lower parts of the y-axis.

3. Look for the peaks that lay above 98% of the maximum value of the result of the second correlation. These peaks correspond to the indices of the synchron pattern within the sequence. The value of 98% is choosen after a number of simulation with different block lengths of the received sequence (i.e. different block sequences) and it is here taken as the threshold value. Figure 3.7 demonstrates the peaks which are ideally placed, between 500 points, and have the unit value of one.

   In order to give more certainty to the correctness of the indices, another verification step is performed, and that is:

4. Divide the block sequence into the "portions of symbols" that correspond to one page length and count the number of peaks (indices) that reside in each "portion". For the case of 10 pages the graphical representation is shown in figure 3.8. The peaks are summed over the whole block sequence and the maximum number of this summation (here 8) gives the number of repetitions of the "true" synchron pattern (since it can appear more times in one block, hence it could also be the "false" pattern) and represents the index of the first synchron pattern that is found in the block sequence. If correct, it will appear in the whole received symbols sequence with identical distances equal to one page length.

In this simulation, the length of one page i.e. one page length is equal to 500 bits, and the block sequence used for examination and finding the pattern is optional, having in mind that longer blocks give more security for the obtained synchron pattern index. Optimal solution is found for the block length equal to 10 times page length.

Symbols extraction: When the first synchron pattern is found correctly, it repeats periodically with the page length as discussed in the previous part. In order to extract the needed symbols between the patterns and to further process them, synchron patterns from the whole received symbol sequence have to be removed. Since the algorithm for finding them operates on blocks that contain 10 pages length (or more) with randomly generated data and appended patterns, the first encountered pattern might not appear exactly at the beginning of the block. Hence, the first page is usually "cut off" and discarded. If the sequential processing is performed (several blocks concatenated), these cut pages have to be considered and stored for the next block iteration. Figure 3.9 highlights this scenario. There is one block sequence with 10 pages presented, where the first cut page is discarded only if it is the first in the whole processed sequence. If that is not the case, it is stored and concatenated with the reminder of the last page of the previous block. This procedure is repeated until the end of the sequence is reached. The last reminder in the last sequence block is discarded also.

After extraction of the symbols, they are stored in a matrix array, where each row corresponds to symbols from one page without the synchron pattern.

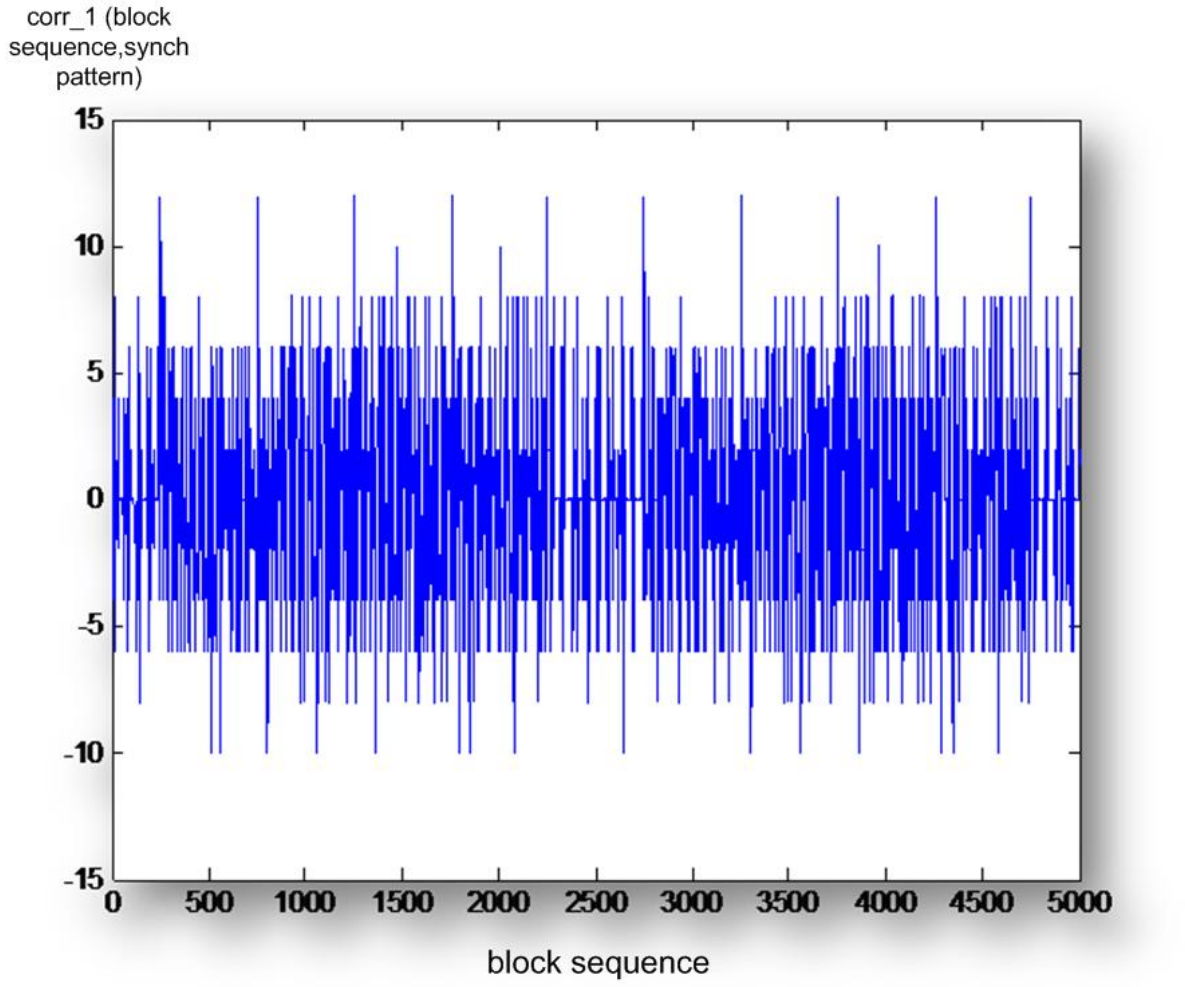

Figure 3.4. Correlation of the block sequence with synchron pattern

Deinterleaver: Deinterleaving is the inverse operation of interleaving, hence one row from the symbols matrix representing one page content is stored into another matrix, called interleaver matrix, with specific dimension according to [21]. It is written row by row and read out of the same matrix column by column. So the symbols in one page are re-ordered/permuted to obtained the original form as before the interleaving.

Decoder: Decoder is implemented as the Viterbi decoder. The symbols from the previous step are decoded employing the Viterbi algorithm as described in Chapter 2 and according to specifications defined in [21]. At the input of the decoder are the symbols from one page and after decoding the corresponding bit sequence exits the decoder, hence the decoding is performed page by page. The length of the bit sequence that exits decoder will be half of the length of the symbol sequence that entered the decoder, since the code rate is equal to $1/2$. In the case of F/NAV message that is simulated, the 488 symbol bits were decoded to 244 data bits. These are then processed to the computation of the CRC checksum and calculation of the error rate.

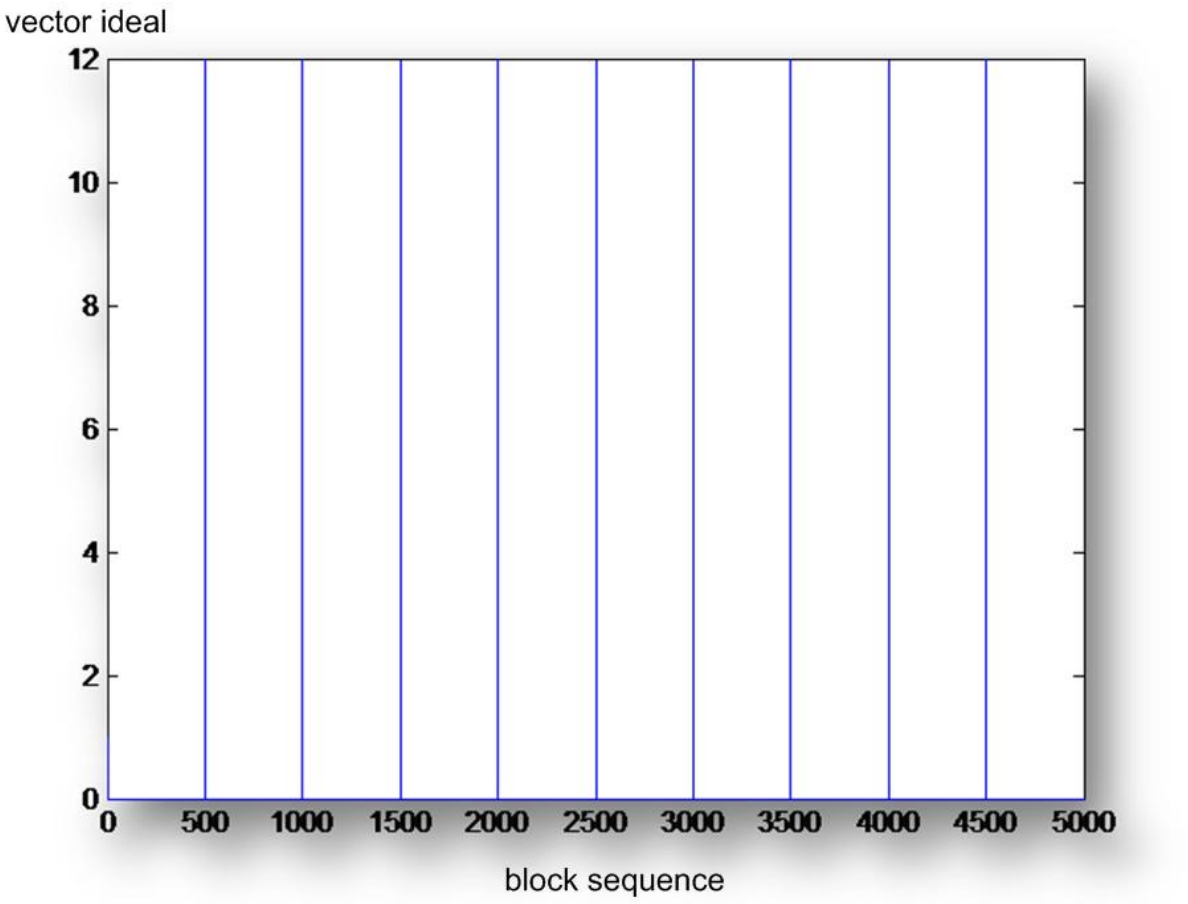

Figure 3.5. Vector of ideally positioned synchron patterns

3.1.4 Test segment

Following tests are executed considering the data bits from the transmitter side, before and after entering encoder as well as from the receiver side, before and after entering the decoder:

1. ***Calculation of CRC checksum and determination of its correctness***

   The calculation of CRC is done in the same manner as it was generated at the transmission segment, employing the same algorithm. This algorithm is calculated with the determined length of the received sequence, and that is the length of the data bits decoded from one page after subtracting the Tail and CRC bits. These are 6 and 24 bits respectively, and when subtracted from 244 bits, it gives 204 bits that are involved in the calculation of the CRC checksum. Again the reminder of the polynomial division is computed, and it results in a 24 bits long bit sequence. The latter is compared with the corresponding CRC bits of the original data sequence. If they are exactly the same, the CRC flag is set to 'Y', if at least one bit is different, then the flag is set to 'N'.

2. ***Calculation of Bit Error Rate (BER)***

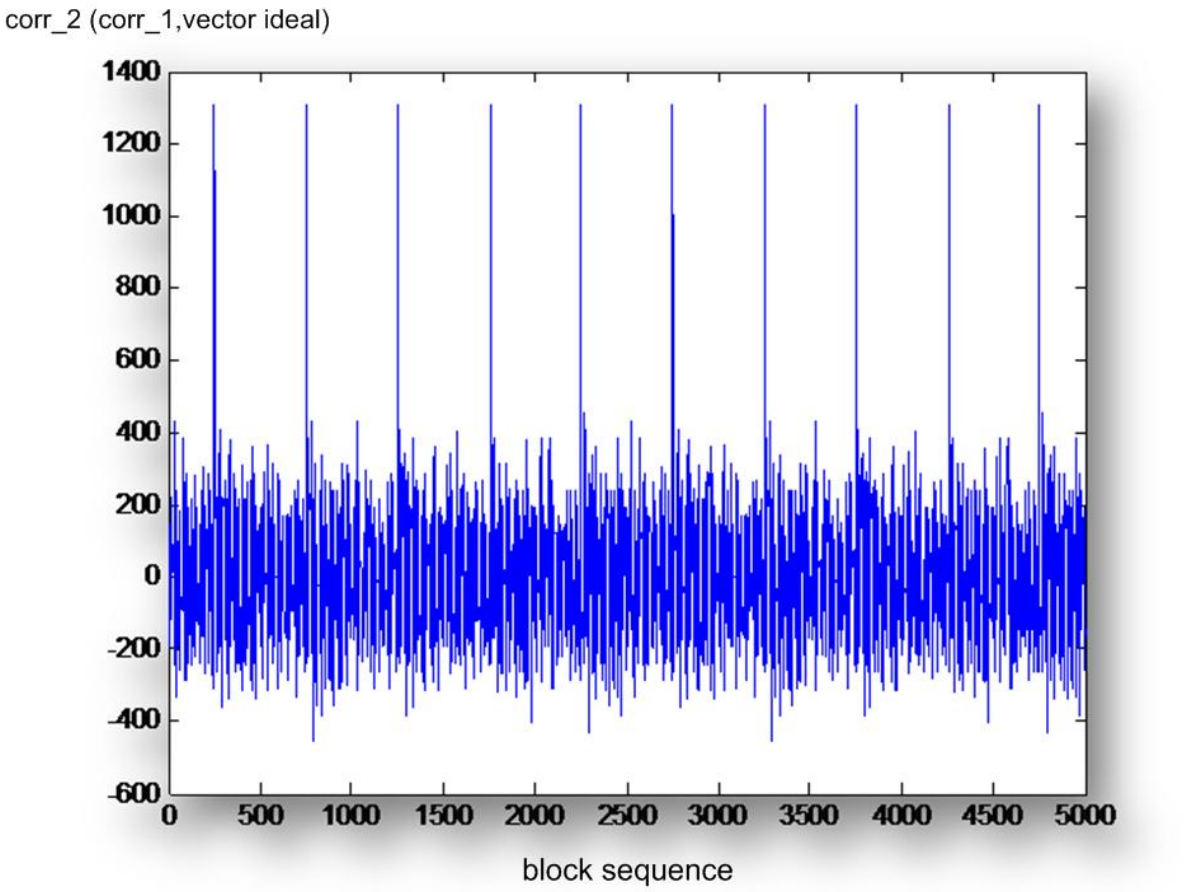

Figure 3.6. Correlation of the first correlation result with the vector of ideally positioned synch patterns

It is done by comparison of the transmitted bits before encoding with the received bits after decoding. The number of bits that differ is summed up and divided by the number of all bits in the transmitted sequence. There are three cases to be considered, namely this calculated value of BER of the coded channel, the calculated value of BER of the uncoded channel (done using the same channel, as usually in simulations) and the theoretical value of BER of the uncoded channel. The calculation of the uncoded BER is performed by comparing the bits that exited the encoder and before they entered the decoder (hereby the name "uncoded" does not correspond exactly the reality, but it is convenient to compute it in that way, since it is computed for the same channel and the decoder performance does not enter into the calculation). This theoretical value is computed using the formula 3.7 (also see Appendix D for details):

$$BER_{uncoded} = 0.5 \cdot erfc\left(\sqrt{\frac{1}{2} \cdot 10^{\frac{E_b/N_0}{10}}}\right) \tag{3.7}$$

The values for each BER for the simulation of F/NAV message sequence $500 \cdot 10^3$ bits long are plotted in graphic 3.10 and they show very good correspondence. The iterations are executed over the Energy per Bit to Noise power spectral density ratio

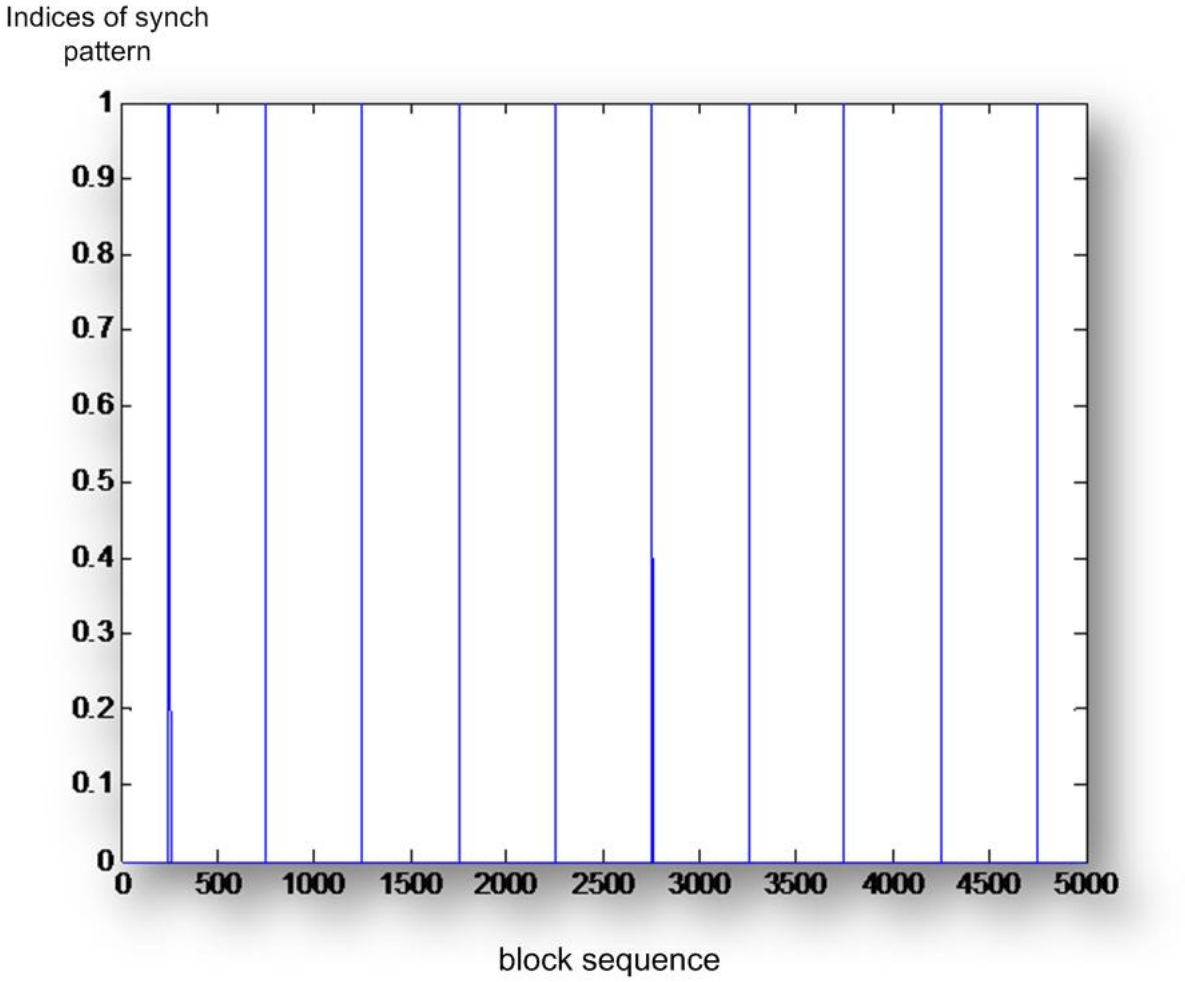

Figure 3.7. Correlation peaks above 98% of the maximum peak value

with values $E_b/N_0 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$ and then compared. The computed theoretical BER values are almost identical with the calculated values for the uncoded case from the simulation for the different values of the E_b/N_0, as seen in the graphic. Green color stands for the obtained BER value from the simulation, and red color for the theoretically computed BER value. As it can be seen in figure 3.10, they overlap completely. The *coding gain*, defined as the reduction in the required E_b/N_0 in a case when coding methods are used compared with the uncoded case for the same BER, coincides with the estimated value as found in various sources [10],[16],[25],[24],[15]. Obtained results are discussed with more details in the next section.

Detailed explanation of the software used for simulation of the navigation channel described above is provided in Appendix A.

3.2 Results and evaluation of BER values

Simulations are executed with various sequence lengths and the Bit Error Rate (BER) computation is performed for various values of E_b/N_0 as it is usually the case in channel sim-

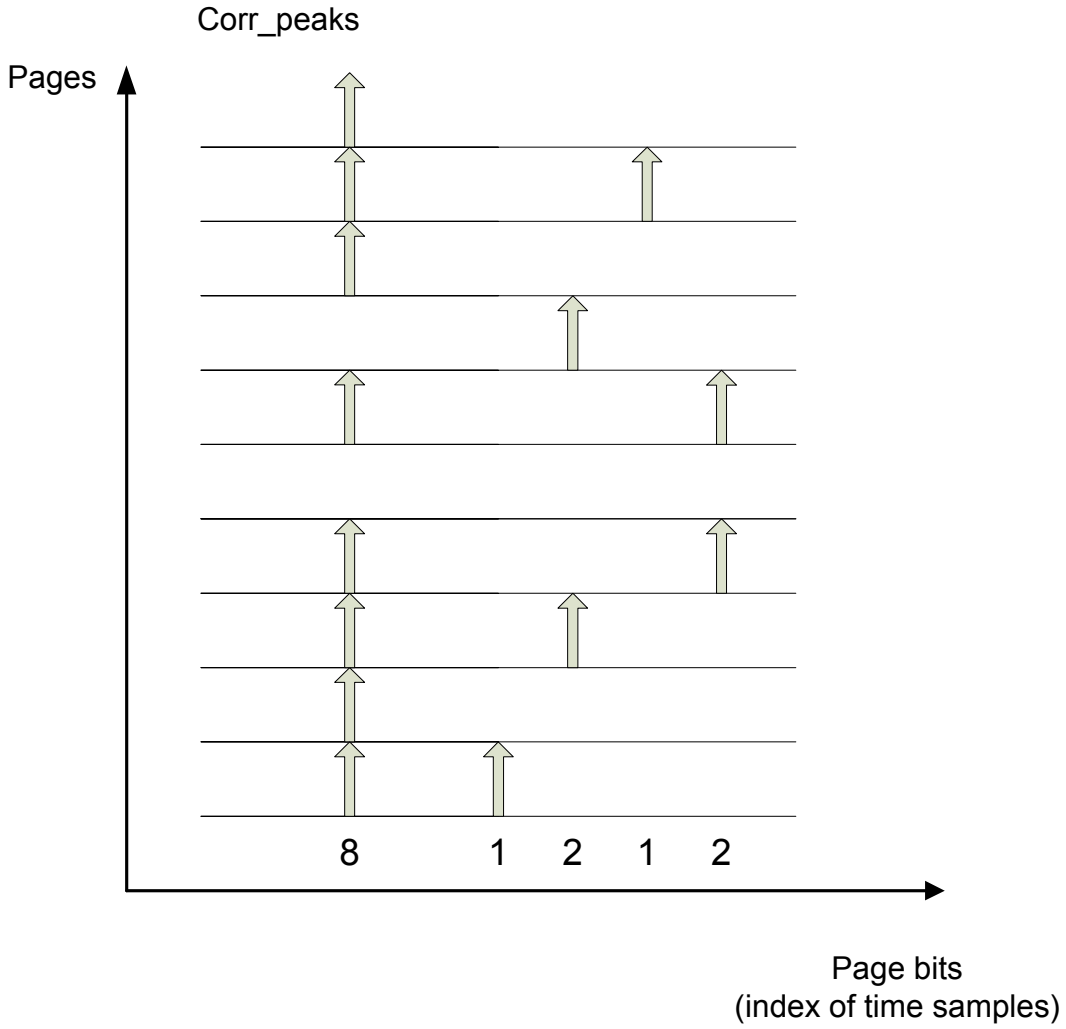

Figure 3.8. Synchron pattern indices

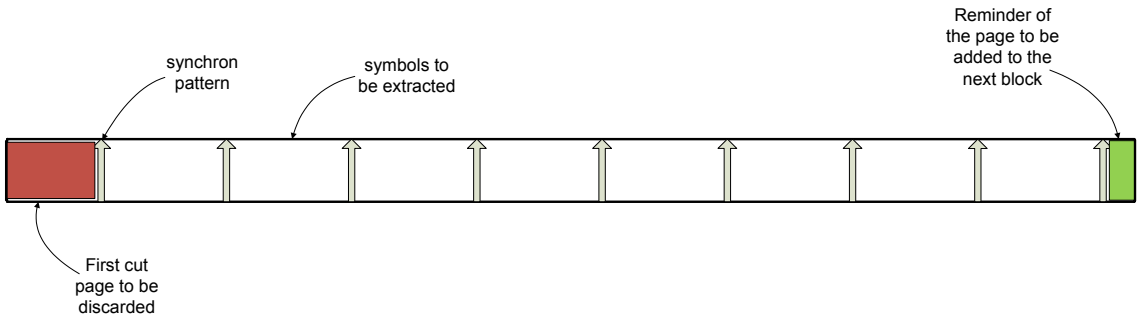

Figure 3.9. Block sequence with 10 pages

ulations. Results are shown in figure 3.10. The simulation is done for values $E_b/N_0 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$ as mentioned before, and the $x-axis$ is actually the Energy per Symbol to Noise power spectral density ratio as defined in the equation 3.5 and which is further equal to $E_s/N_0 = E_b/N_0 - 3dB$.

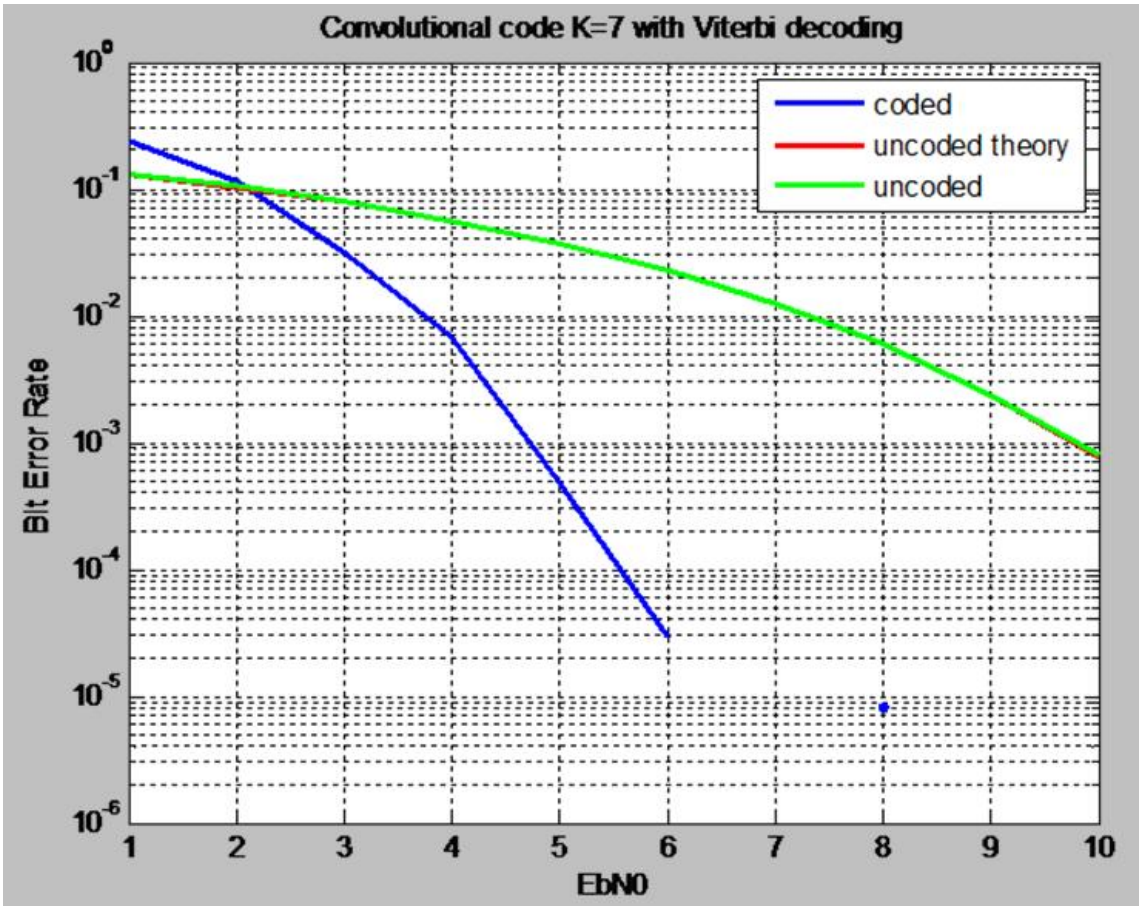

Figure 3.10. Bit Error Rate plot

Blue line refers to the BER for the coded channel, whereby here the hard decision coding is presented. The input bit sequence before the encoding and the output bit sequence after decoding are taken into calculation, as described in the previous section. It approaches the value of 10^{-5} for the minimum BER, which says about the length of the generated sequence for the simulation.

Green line refers to the BER for the uncoded channel. In the simulation, since the channel is coded and the noise on the channel is added randomly, uncoded case is approximated with the following model: symbols that exit the encoder are compared with symbols before they enter the decoder and the value of BER is calculated. It has the same effect as if there was no channel coding implemented, due to the fact that the decoder performance does not enter into calculation. Whether the input bits are encoded or not, does not make any difference here. It is a typical and well-known method used for such channel simulations.

Theoretical value for BER according to equation 3.7 coincide almost perfectly with the estimated value as in [15],[16],[10],[25],[24]. The reason why the coded curve requires a smaller E_b/N_0 for the same value of BER as the uncoded one lays in the fact that the Viterbi error correcting code is used and it has a good performance, hence it is able to correct occurred errors up to some very low values of E_b/N_0. This difference in amount of Energy per bit to Noise power spectral density, actually the reduction in required E_b/N_0 for the same value of BER in the case of coded channel compared to the uncoded one and expressed in

dB is called *Coding gain*.

Although this type of channel coding does enhance the characteristic of the transmission channel by reducing possible errors, it seems that it does not achieve the same performance for very low values of E_b/N_0. This occurs due to the fact that every error-correction code has some determined error-correction capability. When the threshold value of E_b/N_0 is reached, which corresponds to the crosspoint between the coded and uncoded curve (in the figure 3.10 it resides between $E_b/N_0 = 10^{-1}$ and $E_b/N_0 = 10^{-2}$), there are burst errors in transmission that decoder is not able to correct, and its redundant bits are only taking energy from the channel without performing their initial function [10],[16]. Hence in that situation the uncoded BER values are lower than the coded BER values and the performance of the channel is better without the coding. Those burst errors have the length which is greater than the constraint length of the Viterbi encoder (see also Chapter 2).

4 Navigation Signals and Navigation Message Structure

A short overview of Galileo signals, services and types of navigation messages that it will provide to the users that belong to certain user groups is described in this chapter. It also gives an overview over the GIOVE signals previously and currently broadcast.

4.1 Galileo Navigation Signals

Navigation Signals that are envisaged for Galileo use frequencies from Radio Navigation System Service (RNSS) bands, as well as the frequencies from Aeronautical Radio Navigation System (ARNS) bands [13],[21]. These carry out three different composite signals, namely E1,E5 and E6.

Each composite signal generated on board the satellites [21],[8] has a certain carrier frequency and occupies a certain band as defined in table 4.1. Its components, Data and Pilot channel are modulated with a ranging code, multiplexed to form a composite signal and then transmitted in a specific carrier frequency [21]. Signal components that contain navigation data provide different type of navigation services depending on their navigation message content. Those services are Safety of Life Service (SoL), Open Service (OS), Commercial Service (CS) and Public Regulated Service (PRS). They will be described in the next section.

Signal	Carrier Frequency	Bandwidth
E1	1575.420 MHz	24.552 MHz
E6	1278.750 MHz	40.920 MHz
E5	1191.795 MHz	51.150 MHz

Table 4.1. Galileo signals definition [21]

Signal E1 includes three components; two data channels carrying navigation message and one pilot channel containing only the ranging code. Data channels, containing navigation parameters are therefore referred to as navigation signals. Pilot channel provides more precise and robust navigation measurements. Both of them can be used for determination of pseudoranges between satellites and receiver. They all provide different services to users according to table 4.2.

Signal E5 is composed of two navigation signals: E5a, containing data for navigation and timing and E5b, containing navigation and integrity data. They are mapped to services as in the table 4.3 and they are both open service signals.

Signal	Channel	Content	Message Type	Services
E1	A	Data	G/NAV	PRS
	B	Data	I/NAV	OS,CS,SoL
	C	Pilot		

Table 4.2. Signal E1 definition [21]

Signal	Channel	Content	Message Type	Services
E5	E5a	Data	F/NAV	OS
		Pilot		
	E5b	Data	I/NAV	OS,CS,SoL
		Pilot		

Table 4.3. Signal E5 definition [21]

Signal E6 includes three components, two of them used as navigation data signals. They provide different services that can be seen in table 4.4. Both navigation signals are encrypted, either with governmental or with commercial encryption code.

Signal	Channel	Content	Message Type	Services
E6	A	Data	G/NAV	PRS
	B	Data	C/NAV	CS
	C	Pilot		

Table 4.4. Signal E6 definition [21]

4.2 GIOVE Navigation Signals

Test signals that are broadcast from two test satellites, GIOVE-A and GIOVE-B represent the future Galileo signals, regarding the frequencies and modulations used, as well as the data rates. Navigation message parameters are provided in the ground segment as described in Chapter 1; the signals are generated and broadcast over the satellites, and received with a GETR receiver in various stations worldwide [2],[23]. Although GIOVE-A signals should be fully representative of the Galileo Signals, its Navigation Message parameters serve only for experimental purposes; hence its content and structure does not exactly reflect the structure of the Galileo message. Also, GIOVE A spreading codes are different from the Galileo codes, and it can transmit only two signals at a time. GIOVE B is an improvement of GIOVE A [2]. Signal definitions provided by GIOVE A and B [20] are given in table 4.2.

Both test satellites, GIOVE A and GIOVE B can transmit only two signals at a time, and these are either E1-E5 or E1-E6. These signals correspond to the future Galileo signals also named as E1-E5 and E1-E6 (see [20]). In the practical work described in the following sections, the signals E1 and E5 are being evaluated.

Signal	Carrier Frequency	Reference Bandwidth	Message Type
E1	1575.420 MHz	32.736 MHz	–
E5	1191.795 MHz	51.150 MHz	–
E5a	1176.450 MHz	20.460 MHz	F/NAV
E5b	1207.140 MHz	20.460 MHz	I/NAV
E6	1278.750 MHz	40.920 MHz	I/NAV

Table 4.5. Giove signals definition [20]

4.3 Navigation Message

Galileo system employs four types of navigation messages [4],[9] and these are:

F/NAV Freely Accessible Navigation Message Type - its signals are used for provision of Open Service and it is freely available to all public users.

I/NAV Integrity Navigation Message Type - its signals are used for provision of Open Service, Commercial Service and Service of Life. As the name states, this message type contains additional information about Integrity of navigation message i.e. whether the parameters sent in the message are reliable and can be used for positioning and timing.

C/NAV Commercial Access Message Type - provides Commercial Service with higher data rate and with commercial encrypted signals.

G/NAV Governmental Access Message Type - provides Public Regulated Service to specific group of users with governmental encrypted signals.

Correspondingly to different message types, Galileo offers services to different groups of users [4],[9]. There are 4 service types and these are:

OS Open Service - it is free to use and accessible free of charge with a small, low cost receiver. It provides information about positioning, timing and velocity with a certain accuracy. Users will be able to receive combined Galileo and GPS signals. Currently these signals (OS Galileo and C/A code of GPS) are located in different frequency bands. Hence, a dual-frequency receiver can be used for better accuracy due to the corrections of ionospheric errors.

SoL Safety of Life - it is also free of charge, with global high level performance and integrity information provision, in order to increase safety in areas where human safety is critical (aviation, rail, maritime). It provides authentication of a signal, so users can be sure of reception of the right Galileo signal. The SoL signals are also separated in two frequencies and received with dual-frequency receiver.

CS Commercial Service - it is bounded with a fee payment and hence with restricted access. It provides additional information with higher and preciser performance characteristics and value-added applications. It also includes service guarantees, high precise timing and positioning informations. CS signals are encrypted with commercial encryption.

PRS Public Regulated Service is also restricted, used for governmental applications. It provides continuity of service and protection against threats to Galileo signals due to interferences. It improves the continuous availability of signals for the use in security and emergency services, i.e. government-authorized applications. Nevertheless, this service

is controlled by civil institutions, having the signals encrypted with governmental encryption.

Galileo is currently going through its test phase with two test satellites, GIOVE A and GIOVE B broadcasting the future Galileo signals, as already mentioned. Their navigation messages slightly differ from ones foreseen in Galileo, hence it is of importance to emphasize the differences between Galileo message structure and GIOVE test message structure.

4.3.1 Message Generation

Navigation message that is going to be used in Galileo, as well as its version now broadcast from GIOVE A and GIOVE B, is generated both on Earth and on board the satellites. First part of the message, actually the navigation parameters are obtained by the means of measurements of the parameters that GESS stations receive from satellites and proceed to Galileo Control Center (GCC) as described in Chapter 1. It is then uploaded to the satellites, where the other part of the message is generated on-board, and then broadcast to the users. This on-board generated part includes error protections of the data and enhancement of the error correction methods.

The smallest unit that contains navigation data is called Page. The format of one page is defined for each message type in Galileo as well as in GIOVE according to [21],[8] and [20].

On-board generation of page includes the following steps:

1. CRC - Cyclic Redundancy Check
2. FEC - Forward error correction with Tail bits
3. Interleaving of encoded bits

It is done by employing the scheme in table 4.6.

Step 1:		
Navigation data	CRC	Tail
Step 2:		
Navigation data	CRC	Tail
FEC symbols		
Step 3:		
Navigation data	CRC	Tail
FEC symbols		
Interleaving		

Table 4.6. Page generation

After completing these three steps, the synchron pattern is added in the next step (see table 4.7) and one page with symbols is obtained. A certain amount of such pages is concatenated to build a frame that is then to be transmitted in a sequence with other frames generated in the same way.

Step 4: PAGE	
Synchron pattern	Interleaved FEC symbols

Table 4.7. Page

At the reception, these steps are done in a reverse order. Firstly, the synchron pattern is removed and the symbols are deinterleaved. Then they are passed through the Viterbi decoder. After being decoded, navigation data is extracted.

Since the scope of this work is the evaluation of navigation data from F/NAV and I/NAV messages, in further discussion only those two messages will be presented.

Galileo Page Layout: One page will have certain length depending on the message type being transmitted [21]. All pages are generated in the same way.

F/NAV General Message Format

One F/NAV page has the format according to table 4.8

Sync pattern	Symbols	Total Bits
12	488	**500**

Table 4.8. F/NAV Page format

After being received and decoded, the bits are distributed as in table 4.9:

Page Type	Navigation Data	CRC	Tail	Total Bits
6	208	24	6	**244**

Table 4.9. F/NAV Decoded Page

I/NAV General Message Format

One I/NAV page has the format according to table 4.10

After decoding, the page is evaluated differently from F/NAV; it has to be distinguished between two navigation channels, namely E5b and E1B (also see Chapter 5, [8]). Furthermore, when dealing with I/NAV, two consequent messages have to be taken into consideration because they both contain parts of the same navigation stream. The pages are here called words and each word could represent the even or the odd page [21]. For the signal E5b one word will have the format as in table 4.11.

Since it is a nominal page layout, after 1 second the other part of the word is transmitted (either the even or the odd page) and has the format as in table 4.12.

Hence, the first part of one word contains the first part of the data, and the second part of one word the second part, together with External Region Integrity Status (ERIS), Spare, CRC, Region Status (RS).

On the other hand, signal E1-B employs slightly different scheme, where the two parts of the same word are switched and hence transmitted in a different order, according to tables 4.13 and 4.14. The second part of the word is transmitted also 1 second after the first one.

Sync pattern	Symbols	Total Bits
10	240	**250**

Table 4.10. I/NAV Page format

Even/odd=0	Type	Data i(1/2)	Tail	Total Bits
1	1	112	6	**120**

Table 4.11. E5b Word Format - Part 1

GIOVE Page Layout: GIOVE Message is generated in the same way as one that is going to be used in Galileo (and described above), but with different page structure. Yet the page layout is the same for each message type, and it is shown in the table 4.15.

Each field has different length depending on the navigation signal. The lengths of the fields (represented as number of bits) for the signals used for test and evaluation purposes are shown in table 4.16.

According to this scheme and message generation rule from the beginning of this chapter, navigation messages are generated before they are transmitted and the same procedure has to be applied when receiving and evaluating them.

4.3.2 Message Transmission

Navigation message is transmitted in units called frames. Frames are divided into smaller units called sub-frames that are again divided into the smallest units called pages. Each message type has different number of subframes and pages.

As announced in the previous section, only the **E5a** signal that corresponds to **F/NAV** message, and the signals **E5b** and **E1-B** that correspond to **I/NAV** message will be taken into account. Again there is a small difference between the format of the frame in Galileo compared to GIOVE, that will be discussed bellow.

Galileo Frame format: Without going into details, (these can be taken from [21],[8]) there are the two types of evaluated messages.

F/NAV: Free Accessible Navigation message is composed of 12 subframes, each one of 5 pages; hence one frame will contain 60 pages and will be transmitted in 600 seconds. Each page have a duration of 10 seconds.

This pages contain the following parameters: Satellite Vehicle Identification Number (SVID), Signal in Space Accuracy (SISA), Ionospheric correction, Broadcast Group Delay (BGD), Signal Health Status (HS), Galileo System Time (GST), Data Validity Status (DVS), Ephemeris, Almanac as well as GST-UTC [1] and GST-GPS conversion.

I/NAV: One frame is composed of 24 subframes, each one containing 15 Words that are transmitted withing the 30 seconds. Hence, every Word has a duration of 2 seconds, and the whole frame requires 720 seconds for the transmission. The contents of the pages are the same as in F/NAV only with another distribution of bits.

[1] UTC stands for Universal Coordinate Time

Even/odd=1	Type	Data j(2/2)	ERIS	Spare	CRC j	RS	Tail	Total Bits
1	1	16	40	24	24	8	6	**120**

Table 4.12. E5b Word Format - Part 2

Even/odd=1	Type	Data i(2/2)	ERIS	SAR	Spare	CRC i	RS	Tail	Total Bits
1	1	16	40	22	2	24	8	6	**120**

Table 4.13. E1B Word Format - Part 1

Even/odd=0	Type	Data j(1/2)	Tail	Total Bits
1	1	112	6	**120**

Table 4.14. E1B Word Format - Part 2

Sync	Res-1	PGCNT	SNF	NAVDATA	Res-2	CRC	Tail

Table 4.15. GIOVE Page Layout

GIOVE Frame format: Because of the differences in format of the Galileo and GIOVE messages, the latter one need to be considered separately. Here is also given only the short description, further details can be taken from [20].

F/NAV: One Frame consists of 12 Subframes, each one containing 5 Pages as in Galileo. Page numbers are evaluated from Page Counter (PGCNT) field (see table 4.15); it begins counting from one for the first page in the frame, and it is incremented with every new page, until it reaches the number of the pages in one frame, and that is 60. First four pages in one subframe have the same content as first four pages in every other subframe. These are namely the parameters that describe Ionosphere, UTC Conversion, Ephemeris and Clock correction data, as well as the GIOVE A/B System Time, Satellite Health, Broadcast Group Delay, GPS to GIOVE A/B System Time Offset. The fifth page of every subframe contains the Almanac parameters.

I/NAV: There are two Signals for I/NAV: E5b and E1-B. Each of them has distinctive frame format. E5b frame is divided into 25 subframes, each one having 24 pages. That makes 600 pages withing one frame. E1-B frame is composed of 24 subframes, with one subframe containing 25 pages, hence it has 600 pages. As in F/NAV the page numbers are evaluated with the PGCNT field.

Fields/Signals	**E5a**	**E5b**	**E1A**	**E1B**
Length SP	12	10	10	10
Sync Pattern	10110111000	0101100000	0101100000	0101100000
Length Res-1	N/A	1	1	1
Length PGCNT	6	10	10	10
Length SNF	3	3	3	3
Length NAVDATA	217	64	64	64
Length Res-2	N/A	24	24	24
Length CRC	12	12	12	12
Length Tail	6	6	6	6

Table 4.16. GIOVE Page Fields Description

Test analysis is done with GIOVE E5a, E5b and E1-B signals from several test files and represented in Chapter 5. Only few files with simulation of Galileo signals E5a, E5b and L1B are examined; these are also given within the same chapter.

5 Experimental verification

This chapter, i.e. the first part describes the Galileo Experimental Test Receiver (GETR) used for acquirement of the signals from GIOVE satellites [6]. Also the description of the files that the receiver outputs and that are used for the verification of the signals is included. For better understanding of the format and content of these files, some examples are highlighted. In the second part the description of the relevant software developed for testing and processing the GIOVE navigation messages is provided together with the evaluation of the received files.

5.1 GETR inside

GETR is a Galileo/GPS combined test receiver that works in a dual frequency mode and it is able to track all kinds of Galileo and GPS signals. It collects and outputs the following type of data [17],[6]:

- Ranging measurements from GIOVE/Galileo and GPS such as: Pseudorange measurements, Carrier Phase, Doppler and Carrier to Noise ratio $\frac{C}{N_0}$ measurements.
- GPS-based PVT (Positioning Velocity Timing), since only the two GIOVE satellites cannot be used for calculation of those parameters (minimum 4 satellites are required).
- Navigation Symbols (before deinterleaving and decoding)
- Navigation Pages (after deinterleaving and decoding and CRC check)
- Real-time complex correlation peak samples
- Raw IF samples

GETR is able to track up to 7 signals/channels at a time. GIOVE/Galileo signals have Data and Pilot component, hence the GETR will track both of them, with the Pilot component as the default one. However, the tracking mode can be switched. GPS signals do not have the Pilot component, therefore only the Data component is tracked.

In Chapter 4 it was mentioned that the current test satellites, GIOVE A and GIOVE B can only broadcast two signals at the time. These are either $E1+E5$ or $E1+E6$. GIOVE B was emitting the signal combination $E1+E5$ from July 2008 until January 2009. Then it changed to $E1+E6$ from January until March 2009, and returned broadcasting $E1+E5$ again since March 2009 [2].

GIOVE A was emitting both signal combinations $E1+E5$ and $E1+E6$ during its operational life, until July/August 2009. Then it was moved to another (higher) orbit, to release the place for the fully Galileo constellation and prepare the satellite for its end-of-life. Nevertheless, its operational functionality was prolonged and after this manoeuvre, it was able to broadcast $E1+E5$ signals. It was operational until the end of the march 2010.

Navigation data files that are analyzed date from October 2009, and therefore contain only the $E1 + E5$ signal combination.

GETR outputs data in ASCII files. Due to different types of measurements, different types of data files are generated and the appropriate names are given to them. Complete name of each GETR file includes information about the *filename* which is given in a form of date and time when the file was generated and its content. The word that describes the content of the file or at least gives some indication about it, is appended after the underline. Possible files generated by GETR and their filenames are listed in table 5.1 and some example of evaluated files are given within the next section.

filename_meas	In '$@*meas*' messages are all measurements
filename_nav	In '$@*Symb*' messages are raw symbols and in '$@*Page*' messages navigation pages
filename_corr	In '$@*Corr*' messages are correlation peak samples
filename_log	This file contains all error messages from the receiver

Table 5.1. GETR files description [6]

For the purpose of testing the navigation signals, and decoding and evaluation of navigation messages, only two files were needed and used; namely the **filename_nav** and **filename_meas**. They provided all necessary information about the channel number with the corresponding signal component, Pseudo Random Noise number (PRN) which is assigned to each satellite and the received navigation data sequences. Detailed description of each file is given in the GETR User Manual [6].

5.2 Examples of test files

These are two examples of the file types that were generated by GETR and used for examination and verification of navigation messages.

Example 1: File 'generx_091015_165117.meas' Figure 5.1 represents a fragment of the file generated on 15.10.2009. at $16h\,51min\,17sec$ and contains the measurements from GIOVE B satellite.

```
0,53, L1A_1,D,1553,400037,Y, 22940955.530,  -30048.51060, 334.523,     ,43.0,     ,  354.31, 0
1,53,L1BC_2,D,1553,400037,Y, 22940955.144,  216085.27488, 334.422,     ,42.4,     , 1124.60, 0
2,53,L1BC_2,P,1553,400037,Y, 22940955.002,  218275.83850, 334.410,42.3,42.4,     , 1121.52, 0
3,53,     E5a,D,1553,400037,Y, 22940690.985,  233247.66512, 250.125,     ,46.7,     , 1094.08, 0
4,53,     E5a,P,1553,400037,N,  1955218.943,   -4474.60478, 249.540,46.6,46.7,     ,    2.92, 0
6,53,      E5,P,1553,400037,Y, 22940691.311,  214689.86158, 252.819,49.7,46.7,45.7, 1050.32, 0
```

Figure 5.1. Data from 'generx_091015_165117.meas'

First number in each row refers to the channel number. Channels $0-7$ are GIOVE channels (future Galileo channels). The second number is the PRN number of the satellite, where $PRN = 53$ refers to GIOVE B, and $PRN = 51$ to GIOVE A. After this number, the signals and their components, data (D) or pilot (P) are given. These signals can be any of 16 signals broadcast by GIOVE and GPS satellites [6]. The ones that are processed were only the

L1BC_2, L1BC_1 signal components as in the files, which correspond to GIOVE signal E1B; and E5 signal, which comprises E5a and E5b, as in [20].

Channel 6 in *'$@meas'* file stands always for the composite signal E5, actually only for one component E5a (corresponds to F/NAV message). Channel 7 is a virtual channel, hence not present in *'$@meas'*. It is included in channel 6 and represents the E5b component (corresponds to I/NAV message).

These are all parameters that were taken from **"filename_meas"** files. With the knowledge about them, navigation data from files named **"filename_nav"** could be analyzed.

Example 2: File 'generx_091015_165117.nav' In figure 5.2 a fragment of the file which is also generated on 15.10.2009. at $16h\,51min\,17sec$ and should be processed together with the previous file, is presented. It contains navigation symbols and navigation pages.

$@Symb, 1,53,
INAV,1000,051f4eafb87dfa3467e8864ec7851e111f01be147a29fff5ef7c68981ff4805287f37bf21ffea94ff9ac8efdec52e91
7ace57c5f3a7f9843fffb778afc27dae7f8ced8e7fb30521e79d3ed785baccff93d853f869f97fff7ce7ffffffb80007d6e0001fc7
fffff91dfffde1680017e2a0005eaa7e3fffdf187ffffe2a0
$@Page, 1,53, INAV,399256.0,Y, 0,120,20215555555555555540000017500
$@Page, 2,53, INAV,399258.0,Y, 0,120,20615555555555555554000002e940
$@Symb, 2,53,
INAV,1000,47fff80e000001b8800087a5fffa0757ffe8556070000839e00000757fff8055fffe01a000002aa000200b7ffe8791ff
fa12581e000203700000159fffe0267fff806800000b9800082e9fffa0667ffe8316060000805a00000277fff825dfffe03800000
6ae00021587ffe87ddfffa0b581200020040000159fffe03

Figure 5.2. Data from 'generx_091015_165117.nav'

Lines that start with the word '$@*Symb*', which is an indicator for symbols, contain raw symbol sequences. This indicator is followed by the channel number, PRN number and the name of the message type, that can be either F/NAV or I/NAV. Last number in a line before the sequence of hexadecimal numbers represents the length of the symbol sequence proceeding it, which is multiplied by 4, since the symbols are written in hexadecimal form. This number is followed by the sequence of raw symbols. Raw symbols are acquired from the test satellites, where they were generated and contain synchron pattern starting from some index that could be anywhere inside of the symbol sequence. These symbols are extracted and used for further processing with the appropriate algorithm.

Another lines start with the identifier '$@*Page*', which refers to the pages already processed by the hardware components implemented in GETR receiver. This identifier is followed by the channel number, PRN number, name of the message type, transmission Time of Week (TOW), CRC check field (that can be set either to 'Y' or to 'N'), then Viterbi distance field, which is a minimum distance in Viterbi decoder and finally with the number referring to the length of the sequence multiplied with 4, and the actual bit sequence [6]. The length of this sequence is the length of the symbols sequence of one page divided by 2, since the code rate is equal to $1/2$. Here the expression *bit sequence* is used intentionally because these bits are already deinterleaved and decoded (by the mean of hardware implementation of deinterleaver and decoder), therefore they cannot be refer to as symbols anymore. Their purpose within this work is for testing and verification of the implemented algorithm.

The pages are used for a comparison with the results of evaluation of the symbols from the same file. Some pages are discarded right after being received with GETR since they were erroneous, and therefore not useful. That explains the fact that in some files the number

of pages that were processed from raw symbols differs from the number of pages simply extracted from the same file. Also the synchron pattern that is found within the symbol sequence usually cuts the first page after some length, hence the number of actual pages from raw symbols is almost always at least by one smaller than the actual length of the symbols divided by the length of one page.

These and further files are processed and analyzed with the algorithm developed for that purposes, which detailed explanation is given in Appendix B. The results of the tests as well as the brief description of the actual algorithm are provided in the next section. From many files that were received and examined, only the outcome of the four of them is presented bellow, since they are sufficient for the purposes of this work. Furthermore, those files are sufficient in achieving the main goal, which was to test the validity of navigation messages acquired with the test receiver, as well as to test the performance of the developed software. In addition to this, the most important part of the algorithm, to find a synchron pattern in the received sequence of symbols and correctly decode the symbols, was also proven.

5.3 Test environment

Since the receiver part is of interest, the simulation algorithm described in Chapter 3 is restricted only to the reception segment. This had to be adapted to the new environment which was accomplished by including an interface section between the GETR files and the first block in the reception segment, the synchron pattern search. Due to the fact that all files were in the same format, the same procedure of reading the file and extracting data from it could be used for them. This procedure extracts the raw symbols as well as the pages from the same file. As they are written in hexadecimal notation, the conversion to binary format is also included within the routine. The output sequence of raw symbols is then processed using the implemented algorithm.

In figure 5.3 a block diagram of the reception segment is presented, and it can be seen that after the symbols are extracted, the following steps remain unchanged compared to the Chapter 3, where the simulation of a navigation channel was discussed with the deployment of the corresponding software version. At the end of the block chain in a diagram, i.e. after the CRC check is made, the data is stored in form of frames with the structure as defined in [20] for different types of messages.

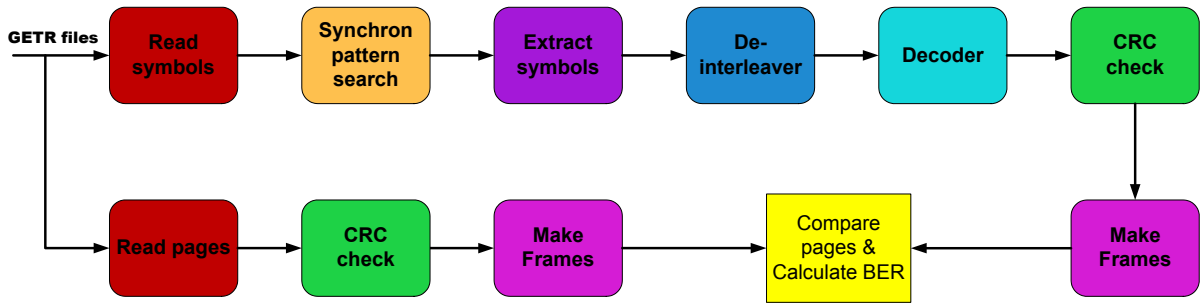

Figure 5.3. Block diagram of reception segment

Following is the description of the blocks from the figure 5.3. As already mentioned, the detailed presentation of each of the functions implemented within this version of the software is provided in Appendix B.

Read symbols is a routine that extracts symbols from GETR files. Example of such file is given in the previous section. The routine reads only the lines starting with '$@*Symb*' word and with the given channel number. Each channel is handled separately, since it refers to one, distinctive signal component. For example, channel 1 could refer to signal component E5a, and channel 2 to E5b. The symbols are written in hexadecimal format, hence they are converted to binary values. Symbols from one channel are stored in one ASCII file in a form of either long sequence of all lines with symbols or an array where each row contains symbols from one '$@*Symb*' line of the same input file.

Read pages does the same as **Read symbols**, only operating on '$@*Page*' lines. Channel numbers and signal components correspond to those from '$@*Symb*' and are extracted in the same way. The page sequences from each channel are also stored in separate ASCII files for a latter use for calculations of the Bit Error Rate (BER).

Synchron pattern search procedure is the same as described in Chapter 3. It is performed over some optimally long portion of the symbol sequence, usually 10 pages length. When the first index is found, it is then updated in every loop iteration, i.e. one page length is added to the previous index to make the new synchron pattern index. During the examination of test data some conclusions concerning the synchron pattern arrangement were made. In some examples, when plotting the correlation function of symbols and the synchron pattern, a huge amount of the highest peaks resided in the negative part of the *y-axis*. That lead to conclusion that the input sequence was inverted, i.e. the whole data sequence was inverted at the time of the reception. Thus after careful investigation of the highest correlation peaks a flag had to be set to indicate whether the symbols are inverted or not.

Extract symbols reads the symbols between two adjacent synchron patterns found in the sequence. The part of the data before the first pattern in the whole sequence is discarded as well as the rest after the last one. There is no sense of taking these symbols into account, since the navigation message which they were carrying is irreversibly lost. In every loop iteration the symbols from one page are extracted and proceeded to deinterleaver and decoder.

Deinterleaver and **Decoder** are implemented exactly as in the simulation of the navigation channel (see again Chapter 3 for details). Here the hard decision decoding is used, hence the corresponding decision flag is set accordingly. Also the soft decision decoding could be used without changing anything in the actual algorithm, only the flag had to be set depending on which type of decoding is chosen.

CRC calculation is performed employing the usual CRC computational algorithm on the received and decoded page. If the CRC checksum is correct, the flag is set to 'Y', otherwise to 'N'. Both, correct and incorrect pages are written in one ASCII file, which has the form of the frame structure: every line starts with the page number, computed from the PGCNT field according to [20] (also look Chapter 4), then the CRC flag is given followed by the navigation data from one page. Every such file contains all pages extracted from only one channel from one GETR file. This is done within the block **Make Frames**. An example of one such ASCII file with the frame structure can be seen in figure 5.4. The pages are written in the order in which they were found in the GETR file with the corresponding page number

that not necessarily begins with one, as in this example. Here the page counter of the first extracted page is equal to 92, hence the pages with lower counter value are not present in the generated ASCII file.

```
090, ,
091, ,
092,Y,000010101010101010101010101010101010101010101010101010101010101010101
093,Y,000010101010101010101010101010101010101010101010101010101010101010101
094,Y,000010101010101010101010101010101010101010101010101010101010101010101
095,Y,000010101010101010101010101010101010101010101010101010101010101010101
096,Y,000000000000000000000000000000000000000000000011111111110000000000010
097,Y,000000000000000000000000000000000000000000000011111111110000000000010
098,Y,000000000000000000000000000000000000000000000011111111110000000000010
099,Y,000010101010101010101010101010101010101010101010101010101010101010101
100,Y,000010101010101010101010101010101010101010101010101010101010101010101
101,Y,000110111010010100110010111111111101000010001110011000000011110101
102,Y,000000000001101111010100001001000111010100111011100101000011011011O
103,Y,000011110000111001101000100101000010010011111001101001100000100010l
104,Y,000011110011011011111000100001111110000011110000001010101010101010l
105,Y,000100001000101100001011100000000000000000000000001110010001100100l
106,Y,000000000000000001101011010000011110101111100010001000000000010101O
107,Y,000010101010101010101010101010101010101010101010101010101010101010101
108,Y,000010101010101010101010101010101010101010101010101010101010101010101
109,Y,000010101010101010101010101010101010101010101010101010101010101010101
110,Y,000010101010101010101010101010101010101010101010101010101010101010101
```

Figure 5.4. Example of the frame structure from the file 'generx_091015_165117_nav.dat'

The same function generates also the same frame structure from the pages extracted from the GETR files with the **Read pages** routine. Both, the extracted pages and the decoded pages from raw symbols enter the **Compare pages and calculate BER** block where pages with the same page number are compared and the differing bits are counted and divided by the length of all pages that were found with the same counter number, in order to calculate the error rate of the appropriate channel. In most of the files that were examined the correspondence between the decoded pages from raw symbols with the extracted pages found in the same GETR file was error-free.

In addition to above commented GETR files from GIOVE satellites, a couple of Galileo test files were also examined. They were simulated with the Galileo Test-Simulator from Spirent [3] and acquired with the same test receiver. The same procedure as in block diagram 5.3 was deployed, except for the part with BER calculations, since there were no test pages included in the files for comparison, only the raw symbols. Corresponding discussion concerning the evaluation of those Galileo test files is also given in the separate section.

5.4 Results evaluation with GIOVE navigation data

This section highlights some examples of files with GIOVE navigation messages that were processed with corresponding algorithm for test purposes. It does not include all files examined in the scope of this work, but rather those that showed some interesting behavior, i.e. some possible scenarios with navigation message at the reception. Not only the data was tested, but also the software performance.

Files that are analyzed were generated during October and November 2009. Evaluated messages are **F/NAV** and **I/NAV**, and the signals mapped to them and supported by GETR are **E5a** for **F/NAV** and **E5b**, **L1BC_1**, **L1BC_2** for the **I/NAV**. Those signals represent either data or pilot signal components, each one transmitted on one separate channel. Another additional signal appears in the files, namely the **L1A_1** and it is here mapped to the **I/NAV** message. That does not actually correspond to Galileo signal specifications [21] and [8], where this signal is mapped to Governmental Navigation Message, **G/NAV**. However it is not encrypted and it does correspond to **E1A** signal according to [20], from which the message format was taken. Therefore the information about the particular parameters for GIOVE signals could be attained.

Another important issue that needs to be considered is that the pilot component involves also the navigation data sequence, which is not foreseen in Galileo. It seems in most of the cases that the channel containing data component of the signal acquired with GETR is copied into the channel reserved for pilot component of the same signal. Thus if both channels exist within the same file, they are identical and only one needs to be processed.

In the following part some files are described, together with the remarks about the channel; especially the cases with inverted navigation symbols were considered.

Example of the file 'generx_091015_163802_nav.dat': This file contains five channels. Channel 0 corresponds to the signal **L1A_1** and **I/NAV** message as commented above. The analysis of this channel lead to the conclusion that all symbols in the symbol sequence were inverted. It further means that the phase of the signal is 180° shifted. The constellation diagram that represents the signal structure can be taken from [21]. The symmetry of this constellation causes a phase ambiguity of 180°. This phase ambiguity has an impact on the symbols and their inversion; in binary sense - one becomes zero and zero becomes one. Consequently, the highest correlation peaks in equal distances that correspond to one page length will now reside in negative part of y-axis, as it can be seen in figure 5.5. Here the negative peaks have the value of 10, since the synchron pattern is 10 bits long. This is one indicator for the inversion of the symbols, if the number of negative peaks with maximum value is larger as the number of the positive ones. Thus those symbols needed to be inverted, before decoding them. Another indicator is the CRC check, which fails in case when received symbols are inverted at the reception without correcting them previously. If there are a lot of pages with false CRC, it could refer to the inversion of the symbols.

When deinterleaving and decoding part are executed, the resulted pages are inserted in the appropriate frame structure. Test pages from GETR files are also evaluated and placed in the same frame structure. The same amount of pages was found. By comparing the pages from both structures and calculation of BER, the CRC is performed and in the case of successful CRC check it was equal to zero, hence no errors were found. Nevertheless there were two pages with false CRC check (equal to 'N') in both frame structures with the same page numbers. This kind of detailed analysis is employed on every GETR file, especially on those where specific results (uncertainties) were obtained.

Channel 1 and Channel 2 of the same file contain the signals **L1BC_2** and **E5a** respectively, and are occupied with pilot components, where the second one seams to be a copy of the Channel 3 which also contains the signal **E5a** but here the data component. Channel 4 and 5 refer to the pilot and to the data component of the same signal, namely the **E5b** for the

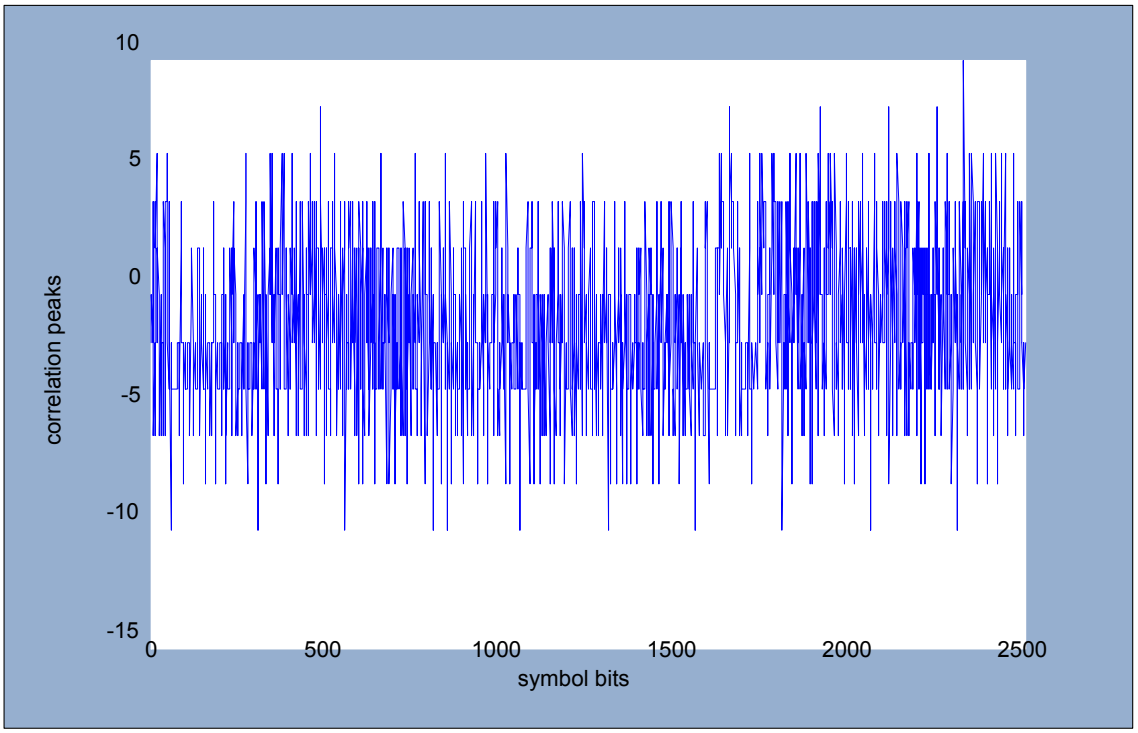

Figure 5.5. Correlation between block symbol sequence and synchron pattern

INAV message. The pages evaluated from those channels coincide perfectly with the test pages from GETR files, hence the BER is everywhere equal to zero.

Example of the file 'gkir042j-k_10e.dat': An interesting example provides this file generated by the GESS station named GKIR, in Kiruna, Sweden. Symbols originated from the signals **E1A** and **E6A** were examined. First one, transmitted on the Channel 0 suffered from inversion of the symbols starting after one determined bit length. That manifested also in a change of the synchron pattern index. Then at some other point, i.e. after some time period the data is inverted again, which resulted in another synchron pattern index. Hence from this second index the original data was received again.

This behavior occurred as a consequence of some external disturbance or interferer on the transmission channel. The receiver had to resynchronize, which resulted in a change of the positions of the synchron pattern within the file due to the symbols that were lost during this process. It could not be corrected before the data entered the last part of the channel, namely the decoding part. There are assumptions that the moment of time when the inversion of the part of the data sequence occurred is related to the activation of the blanking function of the receiver (see [6]). However, detailed description of this phenomena does not fall into the scope of this work.

When the "true" synchron pattern was found, the CRC check was performed over that specific part of the sequence and it showed correct results for every page. Actually the computation of CRC was essential in finding that kind of errors, since it firstly appeared as false over a long part of the sequence, i.e. over lots of pages. This was an indication that those parts of the sequence needed to be investigated carefully and the plots of the correlation peaks yield

to the results observed above. Also the another synchron pattern is found correctly after the inversion of the symbols.

Another signal, **E6A**, assigned to the Channel 2 was also affected with the change of the synchron pattern index after some specific length. However, the symbols were not inverted, only the pattern position within the sequence of the symbols was altered. The resynchronization of the receiver was again the cause of the position change of the synchron pattern index.

5.5 Results evaluation with Galileo Navigation Data

Galileo signals are generated using the RF-constellation simulator from Spirent [3]. It enables controlled performance testing in all conditions; here it is operated in Galileo frequency bands and employs modulation schemes available in [21] as public information. The signals are acquired using the same GETR receiver as in the case of GIOVE signals and the according files are analyzed using the other, additional version of the software adapted for Galileo signals. The following three signals are generated for the following satellites that are foreseen in the future constellation:

- Galileo Satellite nr. 17, broadcasting signal **E1-B** and message type **I/NAV**
- Galileo Satellite nr. 24, broadcasting signal **E5b** and message type **I/NAV**
- Galileo Satellite nr. 24, broadcasting signal **E5a** and message type **F/NAV**

Corresponding files were generated in August 2009. Unfortunately they all contain only short symbol sequences, since the data was recorded over the short periods of time. Only two subframes could be extracted from each of them, usually not comprising all the pages; yet they correspond exactly to the subframe format as defined in [21].

In the file 'tm_binary_log_20090812_0942_rawSymbol_E1B.txt' from the simulation of the satellite nr. 17 for the signal **E1-B** one whole subframe with all 16 data words was found, hence it could be used for the parameter extraction described in the Chapter 6. In the other file were the satellite nr. 24 was simulated i.e. the **E5b** signal, with the name 'tm_binary_log_20090813_1235_rawSymbol_E5B_I.txt' all data was inverted (the same situation as it occurred with some of the GIOVE signals), and after inversion, the CRC check was proven and it resulted in correct values. No full subframe was found, since the structure does not start exactly from the beginning of the first frame. The file with the signal **E5a**, and the name 'tm_binary_log_20090813_1235_rawSymbol_E1A_I.txt', simulating the same satellite nr. 24 provided two full subframes. From some unknown reason, the last pages in each subframe, which should contain Almanac parameters according to [21] appeared as incorrect after the CRC computation.

After the tests with GETR files with both, GIOVE and simulated Galileo navigation messages were performed and the confirmation about the correctly transmitted navigation messages was gained, they are then processed with another software implementation. This software part performs the extraction of navigation parameters from navigation messages, i.e. from the subframes with pages that have the correct CRC checksum. This parameter values are exported in ASCII files in appropriate form and could be used for further computations.

6 Evaluation of navigation parameters

In this chapter the description about the extraction and evaluation of navigation parameters that were processed with the corresponding and modified software is provided. Also the calculation of satellite coordinates from those navigation parameters is performed and the results are presented bellow. Software algorithm that is used is basically the same as described in Chapter 5 for tests and verification of symbols and pages from GETR files, here with some adaptation to the new environment. It is here applied only on symbols with the correct CRC check, in order to extract correct navigation parameters out of them. Results of the execution of this software version are the files with navigation parameters, written in two special formats, called Rinex and Yuma, depending on navigation data which is exported in those formats. Some examples of those files are given within this chapter.

After building a frame structure from navigation symbols, as described in chapter 5 and according to [20] for GIOVE navigation messages and to [21] for simulated Galileo navigation messages, the frames are divided into subframes and the similar structure was generated. Those subframes with pages and page numbers are written in separate ASCII files and can be used for extraction of navigation parameters. First of all, a view words about the output formats used in this work.

6.1 Output in Rinex

RINEX stands for Receiver Independent Exchange Format and it is used for exchange of navigation data, in form of raw data (see [26]). It is defined for three types of ASCII files: Observation data file, Navigation Message file and Meteorological data file.

Since it is the Navigation message which is being evaluated, the obtained parameter values are provided in the Navigation Message file. It contains values of open accessible messages F/NAV and I/NAV in a form similar to GPS messages. In the Header of such Rinex navigation file appears general information about the Rinex version, File Type, Satellite System, Name of Program and Agency creating a file, as well as the information about the date and time when the file was created. This is followed by the Ionospheric Corrections parameters and Time System correction parameters (conversion from GST to UTC time). The other part of the file, Data Record Description contains information about the System time and Clock corrections, Issue of Data (IOD) value, Signal in Space Accuracy value (SISA) and all Ephemeris parameters.

As already highlighted in previous chapters, navigation messages are sent in form of frames, where each frame consists of some amount of subframes, and each subframe comprises some determined number of pages, depending on the message type. In order to produce Rinex file, only one full subframe is necessary in the case of **F/NAV** message and two full subframes in the case of **I/NAV** message, since navigation parameters that are written in Rinex files describe only one satellite, namely the one from which they are broadcast. Therefore even

with a short data records from the receiver, evaluation of maximum two subframes is enough to deliver the desired parameters. One Rinex file looks as in the figure 6.1.

```
     3.00           N                   E                   RINEX VERSION / TYPE
S2R                 Astrium             20091015 163802  UTCPGM / RUN BY / DATE
 GAL   0.0000E+00 -8.2031E-02  2.1942E-02  0.0000E+00       IONOSPHERIC CORR
GAUT  2.7226284146E-05 7.620570841E-13 389120 1553 GSTB    0 TIME SYSTEM CORR
                                                           END OF HEADER
E16 2009 10 11 00 00 00 0.000000000000E+00 0.000000000000E+00 0.000000000000E+00
     4.320000000000E+02 8.646875000000E+01 2.628680923683E-09 2.987405206149E+00
     4.004687070847E-06 1.698527834378E-03 1.445785164833E-05 5.435578960419E+03
     3.888000000000E+05-4.470348358154E-08 2.956294301661E+00 9.126961231232E-08
     9.768762177641E-01 3.662500000000E+01-2.631160825838E+00-5.322007397256E-09
     1.714357124141E-10 2.000000000000E+00 1.553000000000E+03 0.000000000000E+00
     0.000000000000E+00 7.300000000000E+01 0.000000000000E+00 0.000000000000E+00
     3.984300000000E+05 0.000000000000E+00 0.000000000000E+00 0.000000000000E+00
```

Figure 6.1. Rinex from the file 'generx_091015_163802.nav'

It is generated from the file with navigation data named 'generx_091015_163802.nav'. It contains two sections, the upper one is the Header, as already described above. Parameters such as the date and the time of file generation are the same as in the file name (date: 15.10.09 and time: $16h38min02s$), then the name of the company creating the file (Astrium) is given and the other relevant informations (e.g GAL for Galileo system). In addition to that, there are also Ionospheric Correction and Time System correction parameters [26]. Reference time for the calculation of the time system correction polynomial is here given in seconds of the Galileo week (389120), which is broadcast within the navigation message, as well as the corresponding week number (1553), which is also one of the parameters carried within the navigation message. Number 1553 refers to the number of weeks elapsed since the start epoch of the GIOVE system time, which was on Sunday 06.01.1980 at 00:00 UT (Universal Time scale, based on the rotation of the Earth) [20]. Galileo/GIOVE System Time is defined in consistence with the GPS System Time since one of the main goals is that both systems should be interoperable [9].

After the END OF THE HEADER there is a line with the satellite number, here E16 referring to GIOVE B, and after that the date of the first day of the week (one week in Galileo begins at 00:00 UT Sunday) when the file was generated. It is here the first day of the week number 1553. Every proceeding line comprises Ephemeris and other parameters and it is listed as a BROADCAST ORBIT; there are seven of them defined in Rinex. One of those lines contains SISA (Signal In Space Accuracy) value, whereby in GIOVE navigation message it is set to zero (not valid). Other values that are equal to zero, such as satellite clock bias, clock drift and clock drift rate (in broadcast orbit 1) are also used only for experimental purposes and therefore not for positioning of timing services. For further details see [20] and [26].

6.2 Output in Yuma

Yuma format is used for exporting almanac parameters from navigation messages. The format is taken from the GPS system [5], since it is currently the only one suitable that can be used for this purpose. Almanac parameters are given for all satellites in the constellation and therefore every page within one frame contains the values for one satellite, altogether one frame contains parameters from all satellites. Those pages with almanac are divided into slots and the same parameters are repeated in every page, but with different slot number that refers to a different satellite. Some of the pages comprise parameters about orbital planes, and since there are three planes, parameters also appear multiplied within different pages in the frame, but with distinctive values corresponding to each plane. For this reasons, a long data record is needed with at least one frame with all subframes and all pages.

Almanac delivers basically the same information as Ephemeris, but with lower accuracy. Hence, the correction parameters are not present within almanac. There is also one parameter, which does not belong to correction yet it is also not present in almanac part of GIOVE navigation message; and that is the Longitude of Ascending node of Orbit plane at Weekly Epoch, $OMEGA_0$. It should be present in the final Galileo message definition, the reasons why it is missing at GIOVE will not be further discussed. In cases where this parameter is needed for calculations it is taken from Ephemeris data for the corresponding orbit.

Comparison of ephemeris and almanac data showed very good correspondence in the files that were processed. In case of Yuma almanac files, valid parameter values were allocated only in two almanac slots, for each GIOVE satellite. They have the slot number equal to 01 for GIOVE A, and 16 for GIOVE B. These information is contained within the header of every such file, in example in figure 6.2 PRN-01 corresponds to GIOVE A and in example in figure 6.3 PRN-16 corresponds to GIOVE B. They are both taken from the same ASCII file with almanac data. PRN-numbers are incremented for each satellite.

```
******** Week 1553 almanac for PRN-01 ********
ID:                         01
Health:                     1
Eccentricity:               9.2887878418E-004
Time of Almanac(s):         388800
Orbital Inclination(rad):   9.7876855909E-001
Rate of Right Ascen(r/s):   -5.2802199424E-009
SQRT(A)  (m 1/2):           5.451213E+003
Right Ascen at Week(rad):   2.9562943017E+000
Argument of Perigee(rad):   4.7030592613e-002
Mean Anom(rad):             2.2534863122E+000
Af0(s):                     0.0000000000E+000
Af1(s/s):                   0.0000000000E+000
week:                       1553
```

Figure 6.2. Yuma from file 'generx_091015_165117.nav' for GIOVE A

```
******** Week 1553 almanac for PRN-16 ********
ID:                         16
Health:                     1
Eccentricity:               1.6984939575E-003
Time of Almanac(s):         388800
Orbital Inclination(rad):   9.7687505156E-001
Rate of Right Ascen(r/s):   -5.3259361323E-009
SQRT(A)  (m 1/2):           5.435579E+003
Right Ascen at Week(rad):   2.9562943017E+000
Argument of Perigee(rad):   -2.6311609209e+000
Mean Anom(rad):             2.9874051140E+000
Af0(s):                     0.0000000000E+000
Af1(s/s):                   0.0000000000E+000
week:                       1553
```

Figure 6.3. Yuma from file 'generx_091015_165117.nav' for GIOVE B

Both, the ephemeris and almanac data values from GIOVE satellites can be used for computation of satellite coordinates. This computation is performed employing the algorithm taken from [21]. In the case of ephemeris data, it includes 6 Keplerian parameters and the corrections of those as well as the time of ephemeris needed for computations. The eccentric anomaly is computed iteratively, with the initial value taken from [9]. In the case of almanac data, it included also 6 Keplerian parameters, but here without corrections, and the almanac reference time. The initial value for computation of the eccentric anomaly is also taken from [9], and the value for the Longitude of Ascending node of Orbit plane at Weekly Epoch, $OMEGA_0$ from ephemeris data as already mentioned.

These calculations of satellite position within its orbit are performed for different time instants (GST) with ephemeris as well as with almanac values and both results coincide up to some decimal point. When graphically presented, one can clearly see the movement of the satellite in its orbit which has the form of ellipsoid. When compared with the existing values of satellite position from the measurements at the same date and time (here 15.10.2009 at $16h51min17s$), corrected values are obtained, especially in case of the ephemeris data. In figure 6.4 satellite coordinates from ephemeris data for one whole orbital period can be seen, and in figure 6.5 the same coordinates are computed from almanac data.

week	second	X[m]	Y[m]	Z[m]
1553	399600	12409582.794	11359888.125	24310503.039
1553	403200	9357564.475	19013986.803	20582714.852
1553	406800	8758208.929	25118655.144	12799622.319
1553	410400	9368846.453	27866440.151	2489483.244
1553	414000	9170497.697	26773487.672	-8312630.420
1553	417600	6405766.899	22888851.134	-17468969.670
1553	421200	497239.637	18268503.182	-23167920.755
1553	424800	-7606286.727	14970198.508	-24286821.413
1553	428400	-15826781.692	14061740.299	-20612882.207
1553	432000	-21902577.426	15125422.286	-12879298.576
1553	435600	-24422719.283	16494276.390	-2613939.292
1553	439200	-23452774.421	16098904.385	8164786.100
1553	442800	-20456527.071	12512187.733	17342904.265
1553	446400	-17556596.218	5689282.869	23120639.447
1553	450000	-16490452.648	-2951028.415	24360824.967

Figure 6.4. Satellite coordinates from ephemeris data

week	second	X[m]	Y[m]	Z[m]
1553	399600	12411790.187	11357293.421	24310958.162
1553	403200	9360188.631	19011699.080	20584099.013
1553	406800	8761309.380	25116694.722	12801985.269
1553	410400	9372558.867	27865101.712	2492790.087
1553	414000	9175026.983	26773226.581	-8308929.844
1553	417600	6411169.636	22889804.791	-17465936.359
1553	421200	503175.478	18270359.284	-23166527.115
1553	424800	-7600398.938	14972571.254	-24287448.921
1553	428400	-15821400.724	14064504.165	-20615438.693
1553	432000	-21898020.789	15128754.984	-12883559.341
1553	435600	-24419359.303	16498619.940	-2619474.786
1553	439200	-23450905.762	16104798.121	8158957.645
1553	442800	-20456031.841	12519817.736	17338246.349
1553	446400	-17556999.990	5698140.213	23118452.524
1553	450000	-16491368.104	-2941896.001	24361687.270

Figure 6.5. Satellite coordinates from almanac data

7 Conclusion and future work

First part of this work was the development of the software that simulates a navigation channel. The outcome were Bit Error Rate (BER) curves of the channel, which were simulated for coded channel as well as for uncoded channel for the same values of Energy per Bit to Noise power spectral density ratio E_b/N_0. Also the theoretical value of BER for uncoded channel was computed. The resulting curve was almost identical to the simulated value from the channel simulation. Also the BER curve for the coded case coincide with the estimated value according to various sources [16],[10],[25],[24].

The second part of this work uses another software implementation for testing and verification of raw symbols and decoded pages from test receiver, and the comparison of both. Pages obtained from symbols after processing them with the software are equal to the pages that were already decoded with appropriate hardware equipment in the receiver. After successful decoding calculated BER values delivered very good results.

Third and the last part of the work was to export navigation parameters from decoded symbols and verify their values by computing coordinates of the satellite from which the signal was broadcast. Again the obtain results coincided with the ones provided for comparison. Calculation of satellite coordinates was performed with ephemeris parameters as well as with almanac parameters (that usually have lower accuracy) and the obtained values showed very good correspondence.

When four operational satellites within the next phase of Galileo development are launched, the same implemented algorithm for extraction of navigation data and calculation of the positions of the satellites can be used. Future work in this field can be accomplished by expansion of the algorithm to include the calculation of the user position with the parameters from navigation messages. The accurate solution could be possible with minimum four operational satellites in the space.

Bibliography

[1] http://en.wikipedia.org/wiki/Cyclic_redundancy_check.

[2] http://www.giove.esa.

[3] http://www.spirent.com/Positioning-and-Navigation.aspx.

[4] ESA galileo homepage. http://www.esa.int/esaNA/galileo.html.

[5] Yuma almanac format. http://www.navcen.uscg.gov/?pageName=gpsAlmanacs.

[6] *GETR User Manual*, 12 2007.

[7] A.J.Viterbi. *Convolutional Codes and Their Performance in Communication Systems*. IEEE Xplore.

[8] Signal Team EADS Astrium. *Galileo IOV System Support SIS ICD*. 06 2009.

[9] B.Hofmann-Wellenhof, H.Lichtenegger, and E.Wasle. *GNSS Global Navigation Satellite Systems*. SpringerWienNewYork, 2008.

[10] B.Sklar. *Digital Communications:Fundamentals and Applications*. Prentice Hall, 2001.

[11] C.Fleming. *A Tutorial on Convolutional Coding with Viterbi Decoding*, 2006. http://home.netcom.com/~chip.f/viterbi/tutorial.html.

[12] Ch.Langton. *Tutorial 12:Coding and decoding with Convolutional Codes*. http://www.complextoreal.com/chapters/convo.pdf.

[13] E.A.Miret. *Galileo signal in space design*, 5 2005. http://www.galileoic.org/la/files/SignalPresentation_MasterPolito_9thMay2005.pdf.

[14] E.D.Kaplan. *Understanding GPS principles and applications*. Artech House, 1996.

[15] G.C.Clark and J.B.Cain. *Error-Correction Coding for Digital Communications*. Plenium Press, 1981.

[16] J.G.Proakis. *Digital Communications*. McGraw-Hill, 2000.

[17] J.M.Sleewaegen, A.Slimsky, and M.Hollreiser. *Galileo Receivers Experimentation*.

[18] M.Falcone. *Galileo Overall Architecture*. European Space Agency, ESA/ESTEC.

[19] M.Tossaint, S.Binda, J.Hahn, M.Falcone, R.P.Cardeira, and F.Giuntini. *Galileo Initial Validation Step: GIOVE Navigation Message*. `http://www.giove.esa.int/images/userpage/GPCNavmsg.pdf`.

[20] Galileo Project Office. *GIOVE A+B Public SIS ICD*. `http://www.giove.esa.int`, 08 2008.

[21] Galileo Project Office. *Galileo OS SIS ICD*. `http://www.esa.int/esaNA/SEMTHVXEM4E_galileo_0.html`, 02 2010.

[22] P.D.Groves. *Principles of GNSS,Inertial and Multisensor Integrated Navigation Systems*. Artech House, 2008.

[23] R.Piriz, V.Fernandez, M.Cueto (GMV S.A.), P.Tavella, I.Sesia, G.Cerretto (INRiM), J.Hahn, and D.Navarro-Reyes (ESA). *Towards a Galileo Navigation Message*. `http://www.giove.esa.int/images/userpage/PIRIZ_1137_TimeNav07_Piriz.pdf`.

[24] S.Lin and JR. D.J.Costello. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, 2004.

[25] T.K.Moon. *Error Correction Coding: mathematical methods and algorithms*. Wiley, 2005.

[26] University of Bern W. Gurtner, Astronomical Institute. *rinex - the receiver independent exchange format, version 3.00*. `ftp://ftp.unibe.ch/aiub/rinex/rinex300.pdf`, 11 2007.

A Software description of the Navigation Channel Simulation

This program is used for the simulation of a navigation channel. It comprises **Transmitter section**, **Channel section** and **Receiver section**, then **Test section** and **Output section**. Navigation message that is conveyed through the channel is generated with random data, and has the format of *F/NAV* message according to Galileo OS SIS ICD [21]. Block diagram of navigation channel is given in figure A.1.

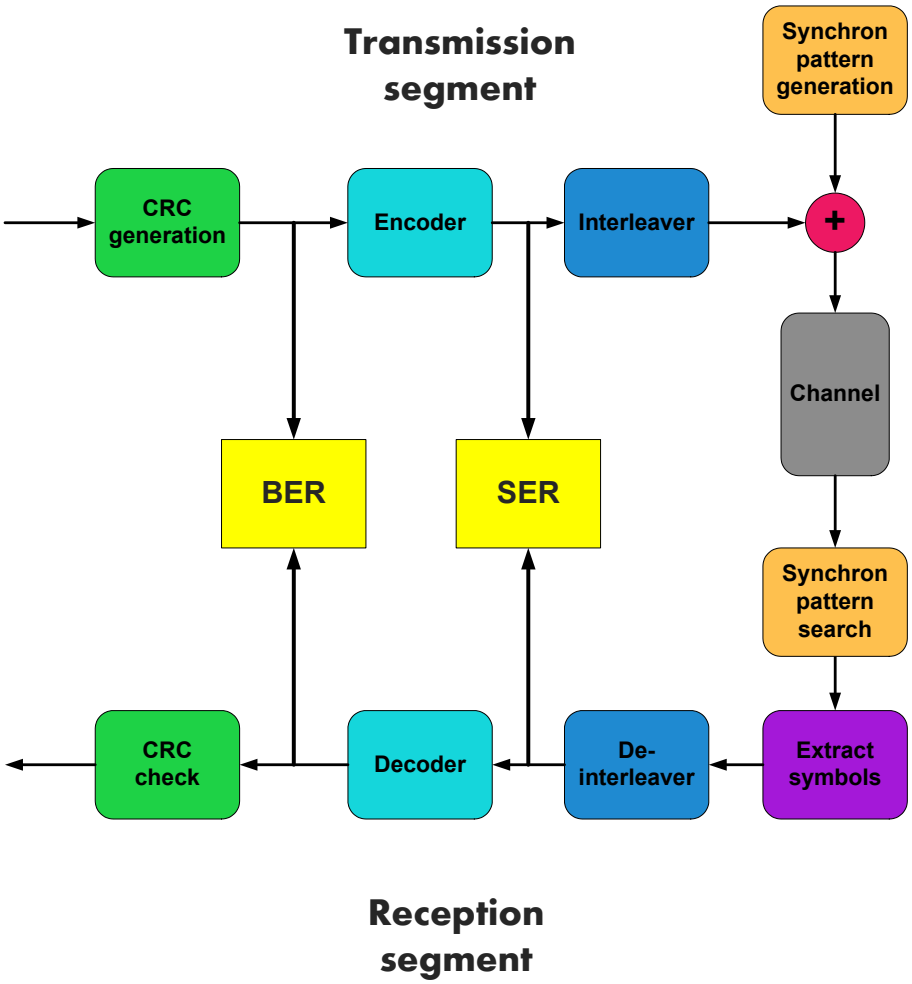

Figure A.1. Block diagram of navigation channel

All programs and functions are written in M-files with the use of Matlab Software, Version 7.8.0 (R2009a). Functions that are implemented and executed in the main program `main.m`

are listed in table A.1 and discussed within this text.

Function name:
msg_structure
crc
encode_conv
interleave
sync_pattern
channel
sync_pattern_search
extract_symbols
deinterleave
decode_vit
generate_tables
convert_i2b
i2b
convert_b2i
b2i
crc_check

Table A.1. List of all functions

Function `[msg] = msg_structure(type)`

This function outputs parameters of different navigation message formats in form of structure in *msg*. Input argument is the type of navigation message and for simulation purposes, it is equal to a string 'fnav'. Input/output arguments are given in table A.2.

Function name	**INPUT**	**OUTPUT**
msg_structure	type - message type	msg.sync_pattern - vector of synchron pattern msg.sp_length - length of synchron pattern msg.page_length - length of one encoded page msg.uncoded_length - length of one uncoded page msg.data_length - length of navigation data field msg.crc_length - length of crc checksum field msg.tail_length - length of tail field msg.tail - vector of tail bits msg.nr_PGinSF_length - nr pages in subframe msg.nr_SFinFR_length - nr subframes in frame msg.interleave_columns - nr columns in interleaver matrix

Table A.2. `msg_structure`

Function `[x_crc] = crc(tx,gen_crc,crc_length)`

This function is used for CRC computation employing the procedure described in [21]. Having *tx* as the sequence of input bits, CRC parity bits are computed as the reminder of the division of *[tx zeros(1,24)]* by generator polynomial *gen_pol*. Since they are both binary

numbers, this operation is performed using the *xor* computation. The CRC checksum will have the length of 24 bits. Input/output arguments are given in table A.3.

Function name	**INPUT**	**OUTPUT**
crc	tx - input bits for crc computation gen_crc - crc generator polynomial crc_length - length of crc field	x_crc - checksum vector

Table A.3. `crc`

Function `[symb_enc] = encode_conv(data,gen_pol)`

This function implements the convolutional encoder as in [21]. Every input bit is encoded to 2 output bits. It is performed as the *xor* operation between the current state of the shift register together with the input bit entering the encoder and each generator polynomial. In every cycle the current state is evaluated to the next state. Input/output parameters are given according to table A.4.

Function name	**INPUT**	**OUTPUT**
encode_conv	data - vector of input bits to encode gen_pol - matrix with generator polynomials, each row corresponds to one polynomial	symb_enc - vector of encoded symbols

Table A.4. `encode_conv`

Function `[symb_interleaved] = interleave(symb_enc,N)`

The function is used to interleave the encoded symbols. Interleaver matrix for the simulated *F/NAV* message is provided in [21]. Symbols are written column by column and read row by row. Input/output parameters are shown in table A.5.

Function name	**INPUT**	**OUTPUT**
interlaeve	symb_enc - vector of encoded symbols N - nr of columns in interleaver matrix	symb_interleaved - vector of interleaved symbols

Table A.5. `interleave`

Function `[symb_trans] = sync_pattern(symb_interleaved,sp)`

This function is used to insert synchron pattern in the symbol sequence. It is inserted randomly, where in one case the pattern is added, and in the other instead of the pattern a vector of zeros in the same length. This is done for test purposes. Input/output parameters are shown in table A.6.

Function name	INPUT	OUTPUT
sync_pattern	symb_interleaved - vector of interleaved symbols sp - synchron pattern	symb_trans - vector of symbols with sync pattern

Table A.6. `sync_pattern`

Function `[symb_soft,symb_hard] = channel(symb_trans,gen_pol,EbN0)`

This function is used to implement an AWGN channel. Symbols are firstly mapped to signals, then the noise is added as the function of E_b/N_0 - Energy per Bit to Noise power spectral density ratio. Afterwards the symbols are either quantized with 3 bits for the soft decision decoder, or with 1 bit for the hard decision decoder. Input/output arguments are given in table A.7.

Function name	INPUT	OUTPUT
channel	symb_trans - vector of symbols gen_pol - matrix with generator polynomials, each row corresponds to one polynomial EbN0 - Energy per bit to Noise power spectral density ratio	symb_soft - vector of symbols after soft decision symb_hard - vector of symbols after hard decision

Table A.7. `channel`

Function `[sp_index] = sync_pattern_search(block_symb,sync_pattern,length_page)`

This function takes one sequence of symbols, that is labeled as *block_symb* and searches for the synchron pattern in it. Input/output parameters are given in table A.8. It comprises three sections:

Function name	INPUT	OUTPUT
sync_pattern_search	block_symb - symbol sequence sync_pattern - synchron pattern length_page - length of one page	sp_index - index of the first synch pattern

Table A.8. `sync_pattern_search`

Parameter definition - Parameters needed for the execution of correlation function that is used in searching algorithm. Vectors with binary values $[0,1]$ are mapped to vectors with values $[-1,1]$ respectively and the zeros are appended to match the length of the symbol sequence with which they will correlate. Vector of ideal positions of the synchron patterns in the examined block sequence is also defined here.

Correlation section - Firstly the correlation between symbol sequence and synchronization pattern is performed with the use of *ifft* and *fft* functions. Multiplication of these two functions of two vectors results in a convolution of both vectors in time domain. Taking one result of the *fft* as conjugated, the outcome is the correlation of time domain signals. Conjugated vector in this case is the synchron pattern. After this first correlation another correlation is

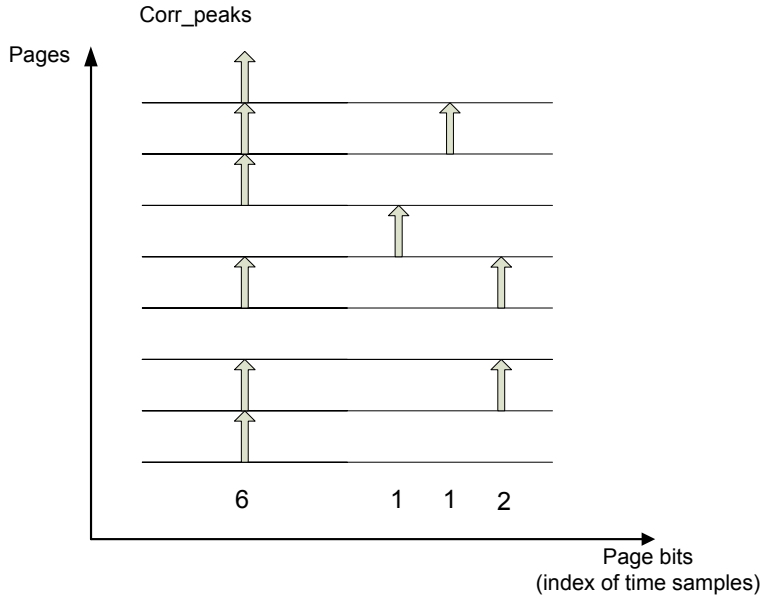

Figure A.2. Synchron pattern search procedure

made, namely the result of the first correlation is correlated again with the vector of ideal position of synchron patterns (spacing of bursts). Resulting values larger than 98% of the maximum value of the peaks are stored into the vector *corr_peaks* and they represent the positions of the synchron patterns found in the examined symbol sequence. This threshold value is chosen after number of simulations with various symbol sequences.

Index extraction - In this part the *corr_peaks* vector is divided into equal lengths, each one corresponding to the length of one page. Since this vector is mostly filled with zeros, the values different from zero will point to bit sequences similar to the pattern. It is assumed that the number of correct positions is greater than the number of incorrect ones, when summed over all pages.
For better understanding one small example is depicted in figure A.2. From vector *corr_peaks* a matrix is formed where each row corresponds to one page. In this example there are 8 pages. This matrix is summed vertically, row by row to build up a vector of the occurrences of the synchron patterns (showed as arrows) within the pages. The numbers bellow correspond to these occurrences. Maximum number is here 6, which means that the synchron pattern appears 6 times correctly at the beginning of each page in the block sequence of 8 pages. Its index in the page corresponds to correct synchron pattern position, and represents the index of the first bit of the first synchron pattern in a given symbol sequence.

Function `[symbols,rest] = extract_symbols(sp_index,block_symb,length_page,length_sp,rest_old)`

This function is used to extract symbols from one block sequence. Synchron pattern is removed and the symbols from each page are stored into one matrix. Since the whole transmitted sequence is divided into blocks, this separation into blocks might also split pages, so the according symbols at the end of each block should be added to the beginning of the next block. Both of them are stored and used in the next call of this function. Input/output parameters are given in table A.9.

Function name	INPUT	OUTPUT
extract_symbols	sp_index - index of the first sync pattern block_symb - vector of one block of symbols length_page - length of one page length_sp - length of synch pattern rest_old - vector of previous rest symbols	symbols - matrix with symbols rest - vector with new rest symbols

Table A.9. `extract_symbols`

Function `[symb_deinterleaved] = deinterleave(symbols,N)`

This function deinterleaves the symbols from each page according to the interleaving scheme employed at the transmitter section. Result of this operation is the vector of encoded symbols as they were before interleaving. Input/output parameters are given in table A.10.

Function name	INPUT	OUTPUT
deinterleave	symbols - vector of symbols to deinterleave N - nr of columns in interleaver matrix	symb_deinterleaved - vector of deinterleaved symbols

Table A.10. `deinterleave`

Function `[data_decoded] = decode_vit(symb_rec,gen_pol,flag_hard_soft)`

This function is used for decoding the encoded symbols employing the Viterbi algorithm. Result of this operation is a bit sequence that has half of the length of the sequence that entered the decoder. Input/output parameters are given in table A.11. The other functions that are implemented and needed for the execution of **`decode_vit`** are **`generate_tables`**, **`convert_i2b`** and **`convert_b2i`**. These are explained bellow as well as the Viterbi algorithm procedure.

Function name	INPUT	OUTPUT
decode_vit	symb_rec - vector of received symbols gen_pol - generator polynomials matrix flag_hard_soft - flag for hard or soft decision	data_decoded - vector of decoded bits

Table A.11. `decode_vit`

Function `[poss_out,next_state,transitions] = generate_tables(gen_pol)`

Three types of tables are necessary for the decoding procedure and they are generated within this function. They can be shown on a small example in figure A.3, where input argument *gen_pol* is equal to $[1,1,1;1,0,1]$. Input/output parameters are given in table A.12.

The first one, **poss_out** contains the encoded pair of bits that are generated for the input bit equal to 0 and for the input bit equal to 1, and for all allowed state transitions. Its size along the rows will always be twice the number of states. First ”vertical half” is the outcome for the input bit 0, and the second ”vertical half” for the input bit 1.

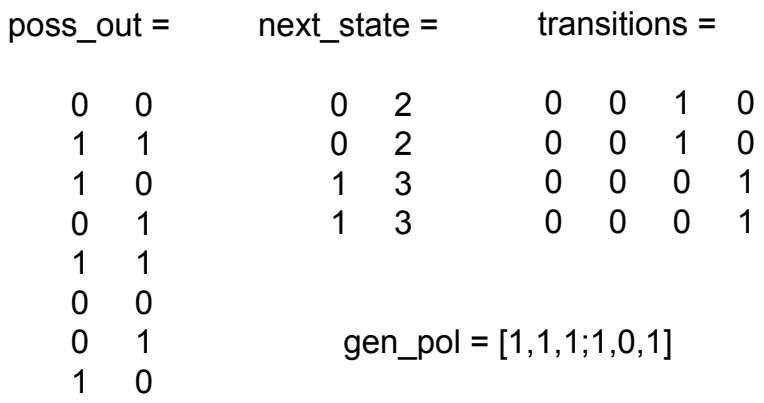

Figure A.3. Tables - output of **`generate_tables`** function

Next_state table contains all possible next states for the given current state and for the given input bit that is encoded. Two different current states and the same input bit (either 1 or 0) can produce the same next state. First column corresponds to input bit 0 and the second to input bit 1.

Transitions table comprises unencoded input bits that imposed all allowed state transitions. Row index correspond to the current state value incremented for one (since the state starts at zero) and column index is the next state value also incremented for one. Input bit 1 in the example causes 4 transitions from current to next state, hence there will be four 1s in the table. Input bit 0 causes also 4 transitions, however the rest of the matrix is also filled with zeros. Hence, if the decoding failed, the ''incorrect" zeros will take the place of unencoded input bits. In successful decoding, where decoded bits are equal to unencoded input bits, they are extracted from this table and correspond to correct zeros and ones.

Function name	**INPUT**	**OUTPUT**
generate_tables	gen_pol - generator polynomials matrix	poss_out - matrix of encoded output bits next_state - next states matrix transitions - matrix of transitions betw.the states

Table A.12. `generate_tables`

Functions `[data_bin] = convert_i2b(data_int)` and `[data_int] = convert_b2i(data_bin)`

In order to carry out binary to integer conversion and vice versa, four functions are implemented, and that are: **`convert_i2b`** converts array of integer values into array of binary values, **`convert_b2i`** converts array of binary values into array of integer values; **`b2i`** makes binary to integer conversion with just one number (not an array) as output, and finally the **`i2b`** accepts only one integer value as input and outputs its binary value with the number of bits assigned to the other input parameter. They are all listed in table A.13.

Now the function **`decode_vit`** can be seen as divided into three sections:

Get parameters - defines and generates the parameters needed for the decoding algorithm. These are the number of states, the number of input bits that are decoded and the path metric

Function name	INPUT	OUTPUT
convert_i2b	data_int - array with integer values	data_bin - array with binary values
convert_b2i	data_bin - array with binary values	data_int - array with integer values
i2b	int - integer number b - number of bits	bin - binary vector
b2i	bin - binary vector	int - integer value

Table A.13. `convert_i2b`, `convert_b2i`, `i2b`, `b2i`

initial value. Path metric matrix is initialized to huge values, called maximum values. This is done because when new metric values are written into the matrix, a minimum has to be chosen. To be sure that it will not come to confusion with the old values, they are assigned to maximum. Function **`generate_tables`** is also executed within this section.

Accumulated path metric - It includes few loops, where the outer one is executed for each time instant, when one bit is encoded. Firstly the pair of encoded bits from the *poss_out* table are compared with the pair of received symbols in the same time instant. In the case of hard decision decoding the Hamming distance is computed, and in the case of soft decision decoding the Euclidian distance. It is then added to the path metric matrix on the corresponding position. This new value represent the new path metric. On Trellis diagram for every time instant there are only two branches that merge in the same node and whose metrics are stored within the path metric matrix. For each node, the minimum of those metric is choosen and then stored in a new matrix, called *Min_metr*. The previous states that yield to this minimum metric are stored in *Prev_state* matrix.

Survivor path - After termination of the outer loop, the survivor path is found in *Prev_state* matrix. This survivor path gives the information about the most probable state transitions that were made, and by employing the *transitions* table from the **`get_tables`** function the decoded bit sequence is found.

At the end of the function, all variables are cleared, since this function is executed a lot of times in the main program, in every iteration for one page of the symbol sequence, therefore it must be assured that old values do not overlap with the new ones.

Function `[crc_flag] = crc_check(data_decoded,gen_crc,crc_length,tail_length)`

This function performs CRC calculation with the decoded data. It executes **`crc`** function, which calculates the checksum that is appended at the end of input data at the transmission segment. After crc calculation the result is compared with the existing crc field in the corresponding page. If they are the same, the flag 'Y' is output, otherwise 'N'. Input/output parameters are given in table A.14.

These function are executed within the main program **`main.m`** and the flow of execution is shown in figure A.4.

Function name	**INPUT**	**OUTPUT**
crc_check	data_decoded - data bits for crc computation gen_crc - crc generator polynomial crc_length - length of crc field tail_length - length of tail field	crc_flag - flag referring to correct or incorrect crc

Table A.14. `crc_check`

Symbols are generated in Transmission segment from the randomly generated input bits. In every loop iteration they are encoded, interleaved and the synchron pattern is added, hence one page is generated. The pages are concatenated to form a sequence of symbols, that is then transmitted over the channel. At the Reception segment, this sequence is divided into blocks of equal lengths. First block is used for finding the synchron pattern. Then the pages without the sync pattern are extracted out of that block, and stored into matrix. Each page is written into one row. In the next loop, pages from the matrix are deinterleaved, decoded and the crc check is performed. This is done until the end of the block, and after completing this step the next block is processed. When the end of the symbol sequence is reached, the obtained data bits from pages are compared with the input bits from the beginning. This procedure is executed for different values of E_b/N_0. It lead to BER (Bit Error Rate)calculation for the coded case. Symbols before decoding and after encoding are also compared, which lead to SER (Symbol Error Rate) calculation that is an abstraction of BER for the uncoded case (see block diagram in figure A.1). The theoretical value of BER uncoded is computed and represented graphically together with the other values over the E_b/N_0 vector.

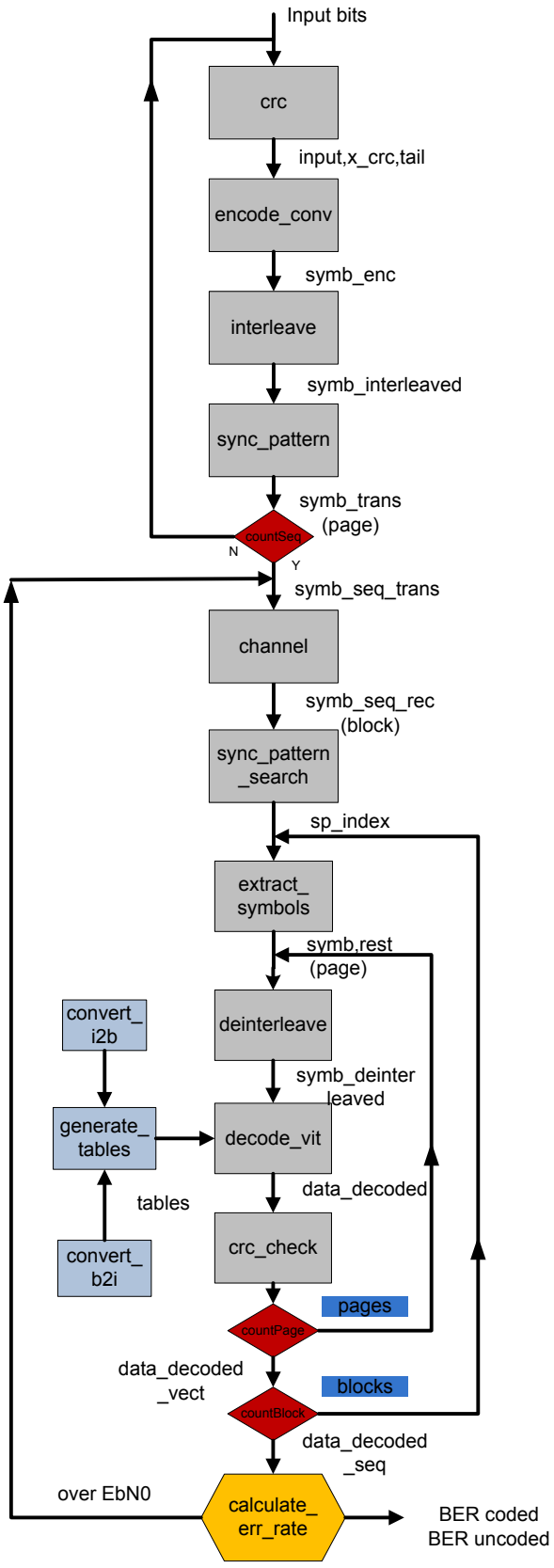

Figure A.4. Flow chart of the main program for sumulation

B Software description of the Tests with GETR pages

This program implements the Reception segment of navigation channel (with Viterbi decoding) and analyzes files with navigation data from the GETR test receiver. It extracts symbols from those files, deinterleaves and decodes them to obtain single pages, and then performs CRC calculation. Since those pages are also contained in the files, they are used for testing - comparison and BER calculation. Block diagram of the program is given in figure B.1. List of functions executed within the main program **`main_getr.m`** can be seen in table B.1.

Function name:
msg_structure
read_getr_symbols
sync_pattern_search
extract_symbols
deinterleave
decode_vit
generate_tables
convert_i2b
i2b
convert_b2i
b2i
crc_check
crc
generate_frames
store_frame_struct
read_getr_pages
calculate_err_rate

Table B.1. List of all functions

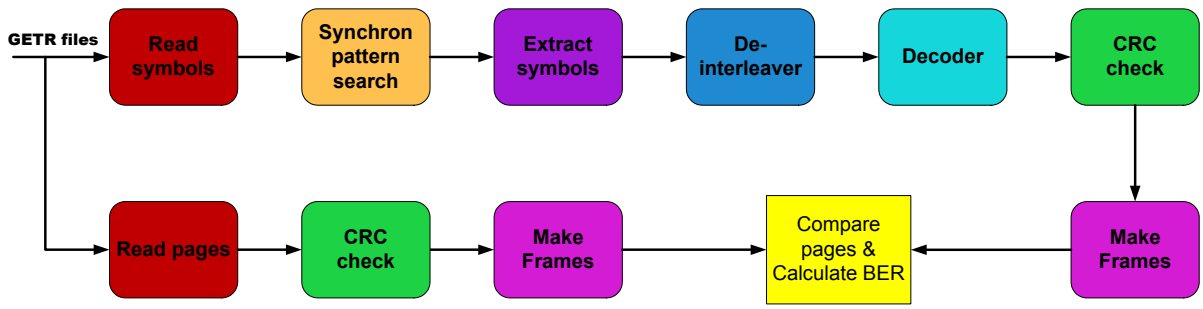

Figure B.1. Block diagram of the reception segment

All functions are written in form of M-files with use of Matlab Software, Version 7.8.0 (R2009a). Description of each function together with its input and output parameters as well as the flowchart of their execution order is presented within this text.

Function `[msg] = msg_structure(type)`

This function outputs parameters of navigation message format in form of structure in *msg*. Input argument is the type of navigation message together with its signal. It is a string, and it can take following values: 'fnavE5a', 'inavE1B', 'inavE5b'. The output parameters within the structure are listed in table B.2 and their values are taken from [20].

Function name	INPUT	OUTPUT
msg_structure	type - message type	msg.sync_pattern - synchron pattern msg.sp_length - length of sync pattern msg.page_length - length of encoded page msg.uncoded_length - length of uncoded page msg.res1_length - length of res-1 field msg.page_cnt_length - length of counter field msg.snf_length - length of snf field msg.navdata_length - length of navdata field msg.res2_length - length of res-2 field msg.crc_length - length of crc checksum msg.tail_length - length of tail field msg.nr_PGinSF_length - nr pages in subframe msg.nr_SFinFR_length - nr subframes in frame msg.interleave_columns - nr columns in interleaver matrix

Table B.2. `msg_structure`

Function `[symb_seq] = read_getr_symbols(ch_nr,filename_read,filename_write)`

Function name	INPUT	OUTPUT
read_getr_symbols	ch_nr - channel number filename_read - name of the file to read filename_write - name of the file to write	symb_seq - extracted symbol sequence

Table B.3. `read_getr_symbols`

This function represents the interface section between GETR file and decoding algorithm. It processes one ASCII file with navigation data from GETR and has input/output parameters according to table B.3. It comprises three sections:

Extraction - Symbols are read only from the lines beginning with '$@*Symb*'. Each line taken from the file should have the corresponding channel number, which is given in the input parameter. The symbol sequence extracted from the corresponding line is given in hexadecimal format. It is stored in a vector inside of the loop every time when it is found in the appropriate line. The procedure is repeated until the end of the file is reached.

Conversion - These hexadecimal symbols are then converted to binary values, 0 and 1. For that purpose one simple function is implemented, called **`i2b`**. It converts integer values to

binary vectors with the determined number of bits (here 4, since every hexadecimal symbol is presented with 4 bits), according to table B.9.

Export - When converted, one long vector containing the binary symbol sequence is written in one ASCII file.

Function `[sp_index,flag_invert] = sync_pattern_search(block_symb,sync_pattern,length_page)`

Function name	INPUT	OUTPUT
sync_pattern_search	block_symb - symbol sequence synch_pattern - synchron pattern length_page - length of one page	sp_index - index of the first synch pattern flag_invert - inversion flag

Table B.4. `sync_pattern_search`

This function takes one sequence of symbols, that is labeled as block_symb and searches for the synchron pattern in it. Input/output parameters are given in table B.4. It comprises three sections:

Parameter definition - Parameters needed for the execution of correlation function that is used in searching algorithm. Vectors with binary values $[0, 1]$ are mapped to vectors with values $[-1, 1]$ respectively and the zeros are appended to match the length of the symbol sequence with which they will correlate. Vector of ideal positions of the synchron patterns in the examined block sequence is also defined here.

Correlation section - Firstly the correlation between symbol sequence and synchronization pattern is performed with the use of *ifft* and *fft* functions. Multiplication of these two functions of two vectors results in a convolution of both vectors in time domain. Taking one result of the *fft* as conjugated, the outcome is the correlation of time domain signals. Conjugated vector in this case is the synchron pattern. After this first correlation another correlation is made, namely the result of the first correlation is correlated again with the vector of ideal position of synchron patterns (spacing of bursts). Resulting values larger than 98% of the maximum value of the peaks are stored into the vector *corr_peaks* and they represent the positions of the synchron patterns found in the examined symbol sequence. This threshold value is chosen after number of simulations with various symbol sequences.

Index extraction - In this part the *corr_peaks* vector is divided into equal lengths, each one corresponding to the length of one page. Since this vector is mostly filled with zeros, the values different from zero will point to bit sequences similar to the pattern. It is assumed that the number of correct positions is greater than the number of incorrect ones, when summed over all pages. For better understanding one small example is depicted in figure B.2. From vector *corr_peaks* a matrix is formed where each row corresponds to one page. In this example there are 8 pages. This matrix is summed vertically, row by row to build up a vector of the occurrences of the synchron patterns (showed as arrows) within the pages. The numbers bellow correspond to these occurrences. Maximum number is here 6, which means that the synchron pattern appears 6 times correctly at the beginning of each page in the block sequence of 8 pages. Its index in the page corresponds to correct synchron pattern position, and represents the index of the first bit of the first synchron pattern in a given symbol sequence.

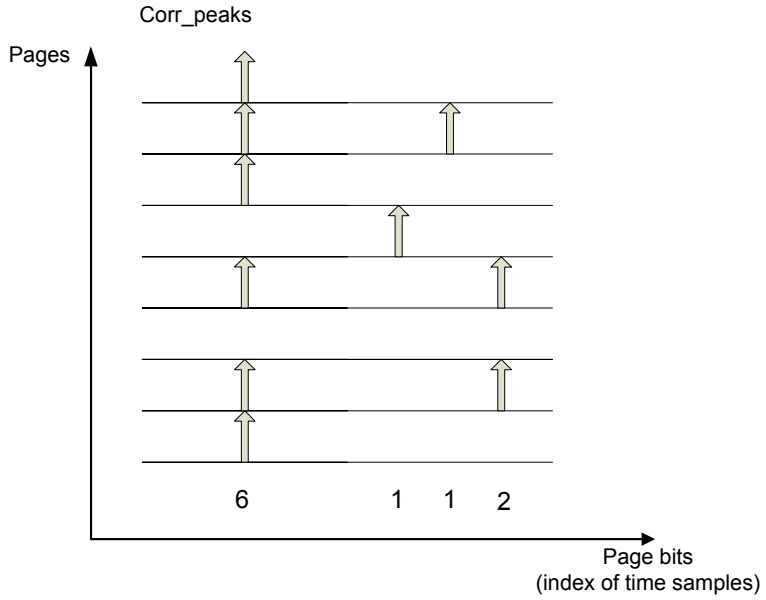

Figure B.2. Synchron pattern search procedure

Since the received symbol sequence can be inverted, there is a flag that is set to one in case when inversion occurs. The condition to set the flag is the number of maximum correlation peaks, which should be greater in the negative part of *y-axis* than in the positive part.

Function `[block_symb_seq] = extract_symbols(symbolsfile,block_length,index,length_sp)`

Function name	INPUT	OUTPUT
extract_symbols	symbolsfile - filename with symbol sequence block_length - length of one block of symbols index - synchron pattern position length_sp - length of synch pattern	block_symb_seq - one block of symbols

Table B.5. `extract_symbols`

This function extracts symbols from the ASCII file produced by execution of **`read_getr_symbols`**. Length of the symbols that are read is determined with the input parameter *block_length*. If the input argument *index* is assigned, then the synchron pattern beginning at this index is removed and the symbols from one page are extracted. If the *index* is not assigned, then only one part of the sequence is read. In both cases character strings from ASCII file are converted to their binary values $[0, 1]$. Input/output parameters are given in table B.5.

Function `[symb_deinterleaved] = deinterleave(symbols,N)`

This function deinterleaves the symbols from each page according to the interleaving scheme employed at the transmitter section. Result of this operation is the vector of encoded symbols as they were before interleaving. Input/output parameters are given in table B.6.

Function name	INPUT	OUTPUT
deinterleave	symbols - vector of symbols to deinterleave N - nr of columns in interleaver matrix	symb_deinterleaved - vector of deinterleaved symbols

Table B.6. `deinterleave`

Function `[data_decoded] = decode_vit(symb_rec,gen_pol,flag_hard_soft)`

This function is used for decoding the encoded symbols employing the Viterbi algorithm. Result of this operation is a bit sequence that has half of the length of the sequence that entered the decoder. Input/output parameters are given in table B.7. The other functions that are implemented and needed for the execution of **`decode_vit`** are **`generate_tables`**, **`convert_i2b`** and **`convert_b2i`**. These are explained bellow as well as the Viterbi algorithm procedure.

Function name	INPUT	OUTPUT
decode_vit	symb_rec - vector of received symbols gen_pol - generator polynomials matrix flag_hard_soft - flag for hard or soft decision	data_decoded - vector of decoded bits

Table B.7. `decode_vit`

Function `[poss_out,next_state,transitions] = generate_tables(gen_pol)`

Three types of tables are necessary for the decoding procedure and they are generated within this function. They can be shown on a small example in figure B.3, where input argument *gen_pol* is equal to $[1,1,1;1,0,1]$. Input/output parameters are given in table B.8.

The first one, **poss_out** contains the encoded pair of bits that are generated for the input bit equal to 0 and for the input bit equal to 1, and for all allowed state transitions. Its size along the rows will always be twice the number of states. First ”vertical half” is the outcome for the input bit 0, and the second ”vertical half” for the input bit 1.

Next_state table contains all possible next states for the given current state and for the given input bit that is encoded. Two different current states and the same input bit (either 1 or 0) can produce the same next state. First column corresponds to input bit 0 and the second to input bit 1.

Transitions table comprises unencoded input bits that imposed all allowed state transitions. Row index correspond to the current state value incremented for one (since the state starts at zero) and column index is the next state value also incremented for one. Input bit 1 in the example causes 4 transitions from current to next state, hence there will be four 1s in the table. Input bit 0 causes also 4 transitions, however the rest of the matrix is also filled with zeros. Hence, if the decoding failed, the ‘’incorrect” zeros will take the place of unencoded input bits. In successful decoding, where decoded bits are equal to unencoded input bits, they are extracted from this table and correspond to correct zeros and ones.

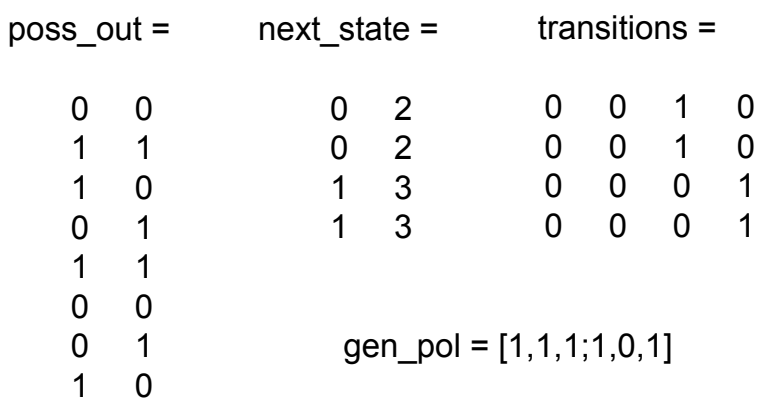

Figure B.3. Tables - output of **`generate_tables`** function

Function name	INPUT	OUTPUT
generate_tables	gen_pol - generator polynomials matrix	poss_out - matrix of encoded output bits next_state - next states matrix transitions - matrix of transitions betw.the states

Table B.8. `generate_tables`

Functions `[data_bin] = convert_i2b(data_int)` and `[data_int] = convert_b2i(data_bin)`

In order to carry out binary to integer conversion and vice versa, four functions are implemented, and that are: **`convert_i2b`** converts array of integer values into array of binary values, **`convert_b2i`** converts array of binary values into array of integer values; **`b2i`** makes binary to integer conversion with just one number (not an array) as output, and finally the **`i2b`** accepts only one integer value as input and outputs its binary value with the number of bits assigned to the other input parameter. They are all listed in table B.9.

Function name	INPUT	OUTPUT
convert_i2b	data_int - array with integer values	data_bin - array with binary values
convert_b2i	data_bin - array with binary values	data_int - array with integer values
i2b	int - integer number b - number of bits	bin - binary vector
b2i	bin - binary vector	int - integer value

Table B.9. `convert_i2b`, `convert_b2i`, `i2b`, `b2i`

Now the function **`decode_vit`** can be seen as divided into three sections:

Get parameters - defines and generates the parameters needed for the decoding algorithm. These are the number of states, the number of input bits that are decoded and the path metric initial value. Path metric matrix is initialized to huge values, called maximum values. This is done because when new metric values are written into the matrix, a minimum has to be

chosen. To be sure that it will not come to confusion with the old values, they are assigned to maximum. Function **`generate_tables`** is also executed within this section.

Accumulated path metric - It includes few loops, where the outer one is executed for each time instant, when one bit is encoded. Firstly the pair of encoded bits from the *poss_out* table are compared with the pair of received symbols in the same time instant. In the case of hard decision decoding the Hamming distance is computed, and in the case of soft decision decoding the Euclidian distance. It is then added to the path metric matrix on the corresponding position. This new value represent the new path metric. On Trellis diagram for every time instant there are only two branches that merge in the same node and whose metrics are stored within the path metric matrix. For each node, the minimum of those metric is choosen and then stored in a new matrix, called *Min_metr*. The previous states that yield to this minimum metric are stored in *Prev_state* matrix.

Survivor path - After termination of the outer loop, the survivor path is found in *Prev_state* matrix. This survivor path gives the information about the most probable state transitions that were made, and by employing the *transitions* table from the **`get_tables`** function the decoded bit sequence is found.

At the end of the function, all variables are cleared, since this function is executed a lot of times in the main program, in every iteration for one page of the symbol sequence, therefore it must be assured that old values do not overlap with the new ones.

Function `[x_crc] = crc(data,crc_length)` and `[crc_flag] = crc_check(data_decoded,crc_length,tail_length)`

This function performs CRC calculation with the decoded data. It executes **`crc`** function, which calculates the checksum that is appended at the end of input data at the transmission segment. After crc calculation the result is compared with the existing crc field in the corresponding page. If they are the same, the flag 'Y' is output, otherwise 'N'. Input/output parameters are given in table B.10.

Function name	INPUT	OUTPUT
crc	data - data bits for crc computation crc_length - length of crc field	x_crc - checksum vector
crc_check	data_decoded - data bits for crc computation crc_length - length of crc field tail_length - length of tail field	crc_flag - flag referring to correct or incorrect crc

Table B.10. `crc` and `crc_check`

Function `[frames] = generate_frames(page,nr_PGinSF,nr_SFinFR)`

This function is used to make a frame structure from all decoded pages from one channel extracted from one GETR file. Pages are stored in form of structure with parameters: *navigation data*, *page number* and *flag*. Each page structure is assigned to one cell and allocated to one frame where it should belong according to its page number. Page number equal to 1 determines the beginning of the new frame. Input/output parameters are shown in table B.11.

Function name	INPUT	OUTPUT
generate_frames	page - symbols from one page nr_PGinSF - nr pages in one subframe nr_SFinFR - nr subframes in one frame	frames - cell structure with frames from one file

Table B.11. `generate_frames`

Function `store_frame_struct(frames,file)`

This function is used to write previously generated frame structure into an ASCII file, where each row contains parameters *page nr*, *crc flag*, *navigation data* from the page structure respectively, separated with commas. It has only input parameters and thay can be seen in table B.12.

Function name	INPUT	OUTPUT
store_frame_struct	-	frames - cell structure with frames file - name of the file to write

Table B.12. `store_frame_struct`

Function `[page_seq] = read_getr_pages(ch_nr,filename_read,filename_write)`

This function does the same as **`read_getr_symbols`** function, only deployed on lines beginning with '$@*Page*' in GETR files. It extracts the symbols from those lines, converts them to binary values and exports them in one file for each channel. Its input/output values are shown in table B.13.

Function name	INPUT	OUTPUT
read_getr_pages	ch_nr - channel number filename_read - name of the file to read filename_write - name of the file to write	page_seq - extracted sequence of pages

Table B.13. `read_getr_pages`

Function `[ber_channel,ber_indx] = calculate_err_rate(frames_symb,frames_page,nr_PGinSF,nr_SFinFR,navdata_length,snf_length)`

This function is used for error rate calculations. It takes the pages obtained from '$@*Symb*' lines out of the frame structure from one channel and the pages obtained from '$@*Page*' lines out of the frame structure from the same channel. It then compares the pages having the same page number bit by bit. Number of different (erroneous) bits divided with the length of the whole sequence from that channel is equal to the Bit Error Rate (BER). Indices of erroneous pages are also given as output. Input/output parameters can be seen in table B.14.

All these functions are executed within the main program **`main_getr.m`** and the flow of execution is presented in figure B.4.

GETR Symbols are first extracted from the file, and stored into a long vector of symbols. Then the first synchron pattern is found in that vector of symbols. With the index of the

Function name	**INPUT**	**OUTPUT**
calculate_err_rate	frames_symb - frames with symbols frames_page - frames with pages nr_PGinSF - nr pages in one subframe nr_SFinFR - nr subframes in one frame navdata_length - length of navdata field snf_length - length of snf field	ber_channel - bit error rate of channel ber_indx - indices of erroneous pages

Table B.14. `calculate_err_rate`

synchron pattern, one loop is performed over the sequence. It takes one page in each iteration, extracts the symbols from it, deinterleaves and decodes them, after which the crc check is performed. Those pages are written in the frame structure and stored in files. Each file contains frames from only one channel.

Since GETR files contain also decoded pages, they are extracted in the same way as symbols and stored into the same frame structure as the pages obtained from symbols. The pages with the same page number are compared and the BER is calculated.

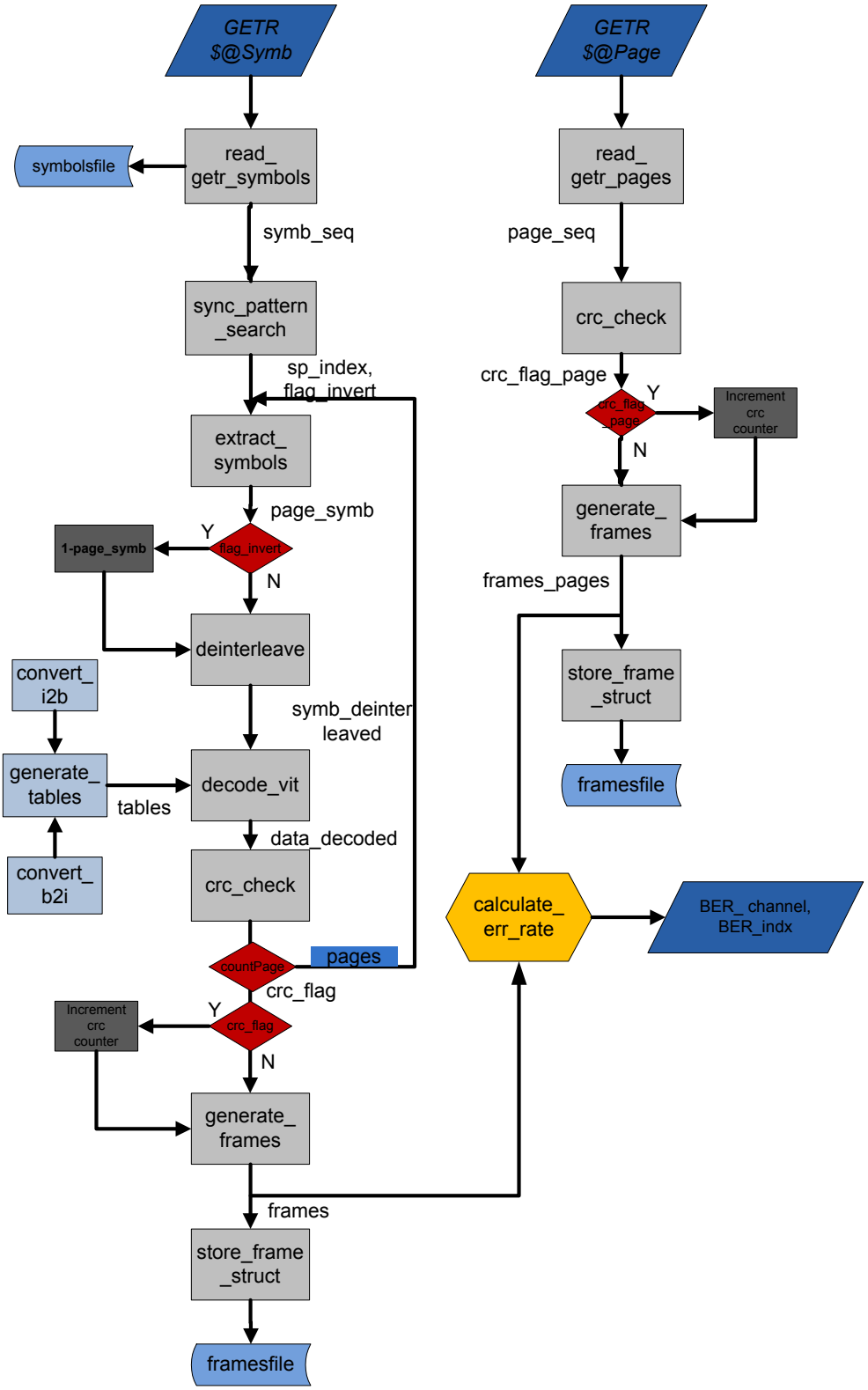

Figure B.4. Flow chart of the main program

C Software description of the Export of RINEX and YUMA files

This program is used to process navigation symbols from GETR files and extract navigation parameters out of them. It exports these parameters in form of ASCII files, that are written either in Rinex or in Yuma format, depending on the type of parameters.

C.1 Main Program

All functions are written in form of M-files with use of Matlab Software, Version 7.8.0 (R2009a). Description of each function together with its input and output parameters as well as the flowchart of their execution order is presented within this text. Main program for one part of the developed software is **`main_getr.m`** and it executes the functions given in the above part of the table C.1. The other main programs are implemented in the form of **`run_rinex.m`** and **`run_yuma.m`** and execute the other three functions in table C.1. Block diagram of the implemented structure is shown in figure C.1.

Function name:
msg_structure
read_getr_symbols
sync_pattern_search
extract_symbols
deinterleave
decode_vit
generate_tables
convert_i2b
i2b
convert_b2i
b2i
crc_check
crc
generate_frames
store_frame_struct
generate_subframes
store_subframes
page_struct_fnav
page_struct_inav
calculate_param

Table C.1. List of all functions

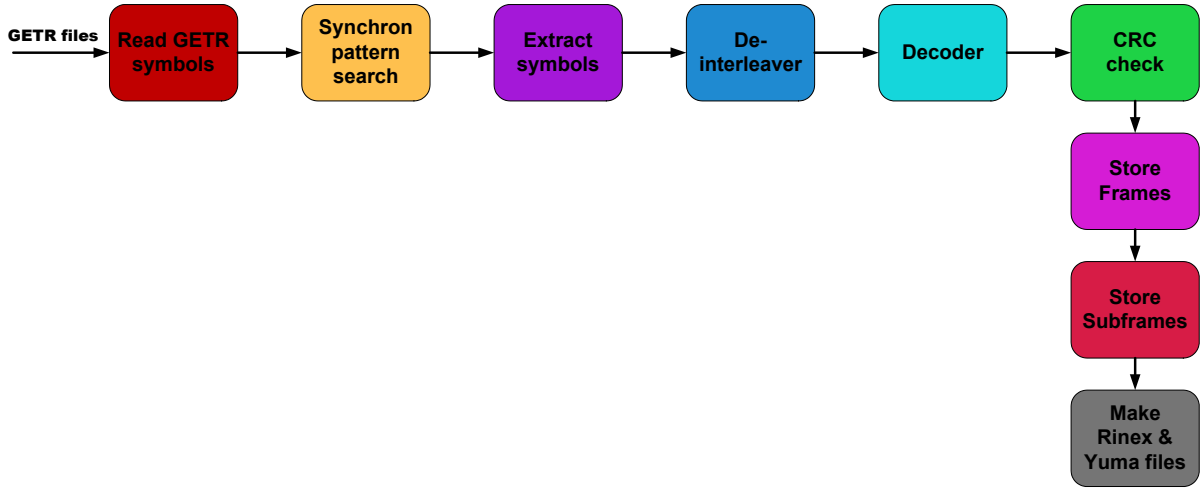

Figure C.1. Block diagram of the reception segment

Function `[msg] = msg_structure(type)`

This function outputs parameters of navigation message format in form of structure in *msg*. Input argument is the type of navigation message together with its signal. It is a string, and it can take following values: 'fnavE5a', 'inavE1B', 'inavE5b'. The output parameters within the structure are listed in table C.2 and their values are taken from GIOVE A+B SIS ICD [20].

Function name	INPUT	OUTPUT
msg_structure	type - message type	msg.sync_pattern - synchron pattern msg.sp_length - length of sync pattern msg.page_length - length of encoded page msg.uncoded_length - length of decoded page msg.res1_length - length of res-1 field msg.page_cnt_length - length of counter field msg.snf_length - length of snf field msg.navdata_length - length of navdata field msg.res2_length - length of res-2 field msg.crc_length - length of crc checksum msg.tail_length - length of tail field msg.nr_PGinSF_length - nr pages in subframe msg.nr_SFinFR_length - nr pages in frame msg.interleave_columns - nr columns in interleaver matrix

Table C.2. `msg_structure`

Function `[symb_seq] = read_getr_symbols(ch_nr,filename_read,filename_write)`

This function represents the interface section between GETR file and decoding algorithm. It processes one ASCII file with navigation data from GETR and has input/output parameters according to table C.3. It comprises three sections:

Extraction - Symbols are read only from the lines beginning with '$@*Symb*'. Each line taken from the file should have the corresponding channel number, which is given in the

Function name	INPUT	OUTPUT
read_getr_symbols	ch_nr - channel number filename_read - name of the file to read filename_write - name of the file to write	symb_seq - extracted symbol sequence

Table C.3. `read_getr_symbols`

input parameter. The symbol sequence extracted from the corresponding line is given in hexadecimal format. It is stored in a vector inside of the loop every time when it is found in the appropriate line. The procedure is repeated until the end of the file is reached.

Conversion - These hexadecimal symbols are then converted to binary values, 0 and 1. For that purpose one simple function is implemented, called **`i2b`**. It converts integer values to binary vectors with the determined number of bits (here 4, since every hexadecimal symbol is presented with 4 bits), according to table B.9.

Export - When converted, one long vector containing the binary symbol sequence is written in one ASCII file.

Function `[sp_index,flag_invert] = sync_pattern_search(block_symb,sync_pattern,length_page)`

Function name	INPUT	OUTPUT
sync_pattern_search	block_symb - symbol sequence synch_pattern - synchron pattern length_page - length of one page	sp_index - index of the first synch pattern in block sequence flag_invert - inversion flag

Table C.4. `sync_pattern_search`

This function takes one sequence of symbols, that is labeled as block_symb and searches for the synchron pattern in it. Input/output parameters are given in table C.4. It comprises three sections:

Parameter definition - Parameters needed for the execution of correlation function that is used in searching algorithm. Vectors with binary values $[0,1]$ are mapped to vectors with values $[-1,1]$ respectively and the zeros are appended to match the length of the symbol sequence with which they will correlate. Vector of ideal positions of the synchron patterns in the examined block sequence is also defined here.

Correlation section - Firstly the correlation between symbol sequence and synchronization pattern is performed with the use of *ifft* and *fft* functions. Multiplication of these two functions of two vectors results in a convolution of both vectors in time domain. Taking one result of the *fft* as conjugated, the outcome is the correlation of time domain signals. Conjugated vector in this case is the synchron pattern. After this first correlation another correlation is made, namely the result of the first correlation is correlated again with the vector of ideal position of synchron patterns (spacing of bursts). Resulting values larger than 98% of the maximum value of the peaks are stored into the vector *corr_peaks* and they represent the positions of the synchron patterns found in the examined symbol sequence. This threshold value is chosen after number of simulations with various symbol sequences.

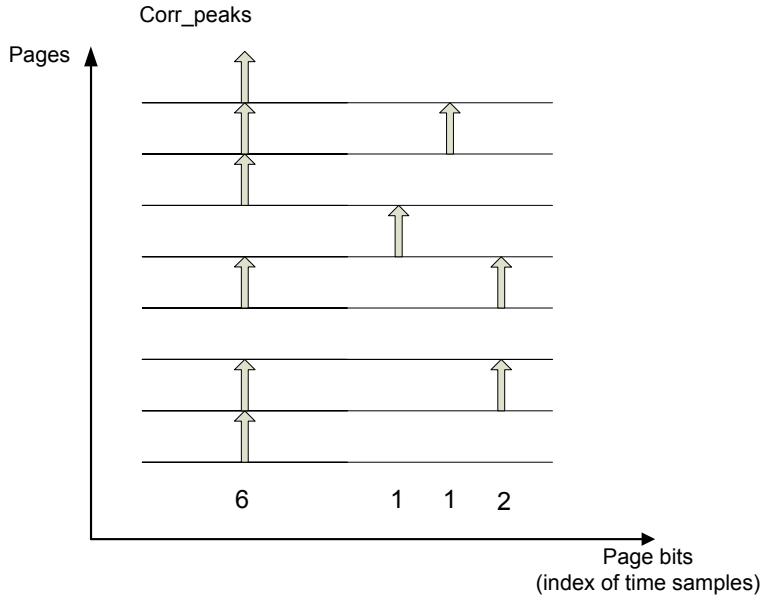

Figure C.2. Synchron pattern search procedure

Index extraction - In this part the *corr_peaks* vector is divided into equal lengths, each one corresponding to the length of one page. Since this vector is mostly filled with zeros, the values different from zero will point to bit sequences similar to the pattern. It is assumed that the number of correct positions is greater than the number of incorrect ones, when summed over all pages. For better understanding one small example is depicted in figure C.2. From vector *corr_peaks* a matrix is formed where each row corresponds to one page. In this example there are 8 pages. This matrix is summed vertically, row by row to build up a vector of the occurrences of the synchron patterns (showed as arrows) within the pages. The numbers bellow correspond to these occurrences. Maximum number is here 6, which means that the synchron pattern appears 6 times correctly at the beginning of each page in the block sequence of 8 pages. Its index in the page corresponds to correct synchron pattern position, and represents the index of the first bit of the first synchron pattern in a given symbol sequence.

Since the received symbol sequence can be inverted, there is a flag that is set to one in case when inversion occurs. The condition to set the flag is the number of maximum correlation peaks, which should be greater in the negative part of *y-axis* than in the positive part.

Function `[block_symb_seq] = extract_symbols(symbolsfile,block_length,index,length_sp)`

Function name	INPUT	OUTPUT
extract_symbols	symbolsfile - filename with symbol sequence block_length - length of one block of symbols index - synchron pattern position length_sp - length of synch pattern	block_symb_seq - one block of symbols

Table C.5. `extract_symbols`

This function extracts symbols from the ASCII file produced by execution of **`read_getr_symbols`**. Length of the symbols that are read is determined with the input parameter *block_length*. If

the input argument *index* is assigned, then the synchron pattern beginning at this index is removed and the symbols from one page are extracted. If the *index* is not assigned, then only one part of the sequence is read. In both cases character strings from ASCII file are converted to their binary values $[0, 1]$. Input/output parameters are given in table C.5.

Function `[symb_deinterleaved] = deinterleave(symbols,N)`

This function deinterleaves the symbols from each page according to the interleaving scheme employed at the transmitter section. Result of this operation is the vector of encoded symbols as they were before interleaving. Input/output parameters are given in table C.6.

Function name	INPUT	OUTPUT
deinterleave	symbols - vector of symbols to deinterleave N - number of columns in interleaver matrix	symb_deinterleaved - vector of deinterleaved symbols

Table C.6. `deinterleave`

Function `[data_decoded] = decode_vit(symb_rec,gen_pol,flag_hard_soft)`

This function is used for decoding the encoded symbols employing the Viterbi algorithm. Result of this operation is a bit sequence that has half of the length of the sequence that entered the decoder. Input/output parameters are given in table C.7. The other functions that are implemented and needed for the execution of **`decode_vit`** are **`generate_tables`**, **`convert_i2b`** and **`convert_b2i`**. These are explained bellow as well as the Viterbi algorithm procedure.

Function name	INPUT	OUTPUT
decode_vit	symb_rec - vector of received symbols gen_pol - generator polynomials matrix flag_hard_soft - flag for hard or soft decision	data_decoded - vector of decoded bits

Table C.7. `decode_vit`

Function `[poss_out,next_state,transitions] = generate_tables(gen_pol)`

Three types of tables are necessary for the decoding procedure and they are generated within this function. They can be shown on a small example in figure C.3, where input argument *gen_pol* is equal to $[1, 1, 1; 1, 0, 1]$. Input/output parameters are given in table C.8.

The first one, **poss_out** contains the encoded pair of bits that are generated for the input bit equal to 0 and for the input bit equal to 1, and for all allowed state transitions. Its size along the rows will always be twice the number of states. First "vertical half" is the outcome for the input bit 0, and the second "vertical half" for the input bit 1.

Next_state table contains all possible next states for the given current state and for the given input bit that is encoded. Two different current states and the same input bit (either 1 or 0) can produce the same next state. First column corresponds to input bit 0 and the second to input bit 1.

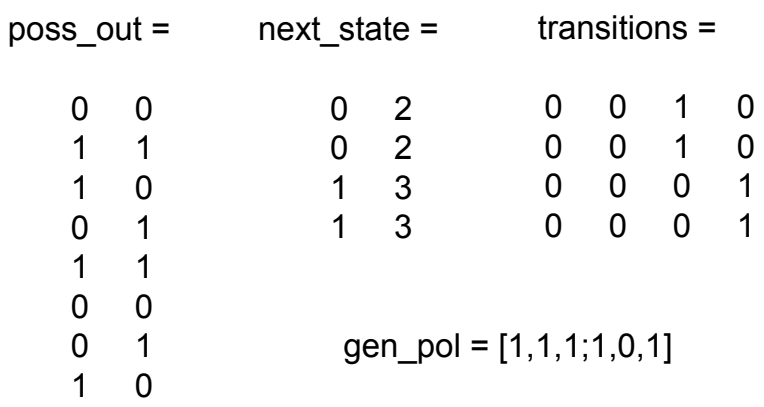

Figure C.3. Tables - output of **`generate_tables`** function

Transitions table comprises unencoded input bits that imposed all allowed state transitions. Row index correspond to the current state value incremented for one (since the state starts at zero) and column index is the next state value also incremented for one. Input bit 1 in the example causes 4 transitions from current to next state, hence there will be four 1s in the table. Input bit 0 causes also 4 transitions, however the rest of the matrix is also filled with zeros. Hence, if the decoding failed, the ''incorrect" zeros will take the place of unencoded input bits. In successful decoding, where decoded bits are equal to unencoded input bits, they are extracted from this table and correspond to correct zeros and ones.

Function name	INPUT	OUTPUT
generate_tables	gen_pol - generator polynomials matrix	poss_out - matrix of encoded output bits next_state - next states matrix transitions - matrix of transitions betw.the states

Table C.8. `generate_tables`

Functions `[data_bin] = convert_i2b(data_int)` and `[data_int] = convert_b2i(data_bin)`

In order to carry out binary to integer conversion and vice versa, four functions are implemented, and that are: **`convert_i2b`** converts array of integer values into array of binary values, **`convert_b2i`** converts array of binary values into array of integer values; **`b2i`** makes binary to integer conversion with just one number (not an array) as output, and finally the **`i2b`** accepts only one integer value as input and outputs its binary value with the number of bits assigned to the other input parameter. They are all listed in table C.9.

Now the function **`decode_vit`** can be seen as divided into three sections:

Get parameters - defines and generates the parameters needed for the decoding algorithm. These are the number of states, the number of input bits that are decoded and the path metric initial value. Path metric matrix is initialized to huge values, called maximum values. This is done because when new metric values are written into the matrix, a minimum has to be chosen. To be sure that it will not come to confusion with the old values, they are assigned to maximum. Function **`generate_tables`** is also executed within this section.

Function name	INPUT	OUTPUT
convert_i2b	data_int - array with integer values	data_bin - array with binary values
convert_b2i	data_bin - array with binary values	data_int - array with integer values
i2b	int - integer number b - number of bits	bin - binary vector
b2i	bin - binary vector	int - integer value

Table C.9. `convert_i2b`, `convert_b2i`, `i2b`, `b2i`

Accumulated path metric - It includes few loops, where the outer one is executed for each time instant, when one bit is encoded. Firstly the pair of encoded bits from the *poss_out* table are compared with the pair of received symbols in the same time instant. In the case of hard decision decoding the Hamming distance is computed, and in the case of soft decision decoding the Euclidian distance. It is then added to the path metric matrix on the corresponding position. This new value represent the new path metric. On Trellis diagram for every time instant there are only two branches that merge in the same node and whose metrics are stored within the path metric matrix. For each node, the minimum of those metric is choosen and then stored in a new matrix, called *Min_metr*. The previous states that yield to this minimum metric are stored in *Prev_state* matrix.

Survivor path - After termination of the outer loop, the survivor path is found in *Prev_state* matrix. This survivor path gives the information about the most probable state transitions that were made, and by employing the *transitions* table from the **`get_tables`** function the decoded bit sequence is found.

At the end of the function, all variables are cleared, since this function is executed a lot of times in the main program, in every iteration for one page of the symbol sequence, therefore it must be assured that old values do not overlap with the new ones.

Function `[x_crc] = crc(data,crc_length)` and `crc_flag = crc_check(data_decoded,crc_length,tail_length)`

This function performs CRC calculation with the decoded data. It executes **`crc`** function, which calculates the checksum that is appended at the end of input data at the transmission segment. After crc calculation the result is compared with the existing crc field in the corresponding page. If they are the same, the flag 'Y' is output, otherwise 'N'. Input/output parameters are given in table C.10.

Function name	INPUT	OUTPUT
crc	data - data bits for crc computation crc_length - length of crc field	x_crc - checksum vector
crc_check	data_decoded - data bits for crc computation crc_length - length of crc field tail_length - length of tail field	crc_flag - flag referring to correct or incorrect crc

Table C.10. `crc` and `crc_check`

Function `[frames] = generate_frames(page,nr_PGinSF,nr_SFinFR)`

This function is used to make a frame structure from all decoded pages from one channel extracted from one GETR file. Pages are stored in form of structure with parameters: *navigation data*, *page number* and *flag*. Each page structure is assigned to one cell and allocated to one frame where it should belong according to its page number. Page number equal to 1 determines the beginning of the new frame. Input/output parameters are shown in table C.11.

Function name	INPUT	OUTPUT
generate_frames	page - symbols from one page nr_PGinSF - nr pages in one subframe nr_SFinFR - nr subframes in one frame	frames - cell structure with frames from one file

Table C.11. `generate_frames`

Function `store_frame_struct(frames,file)`

This function is used to write previously generated frame structure into an ASCII file, where each row contains parameters *page nr*, *crc flag*, *navigation data* from the page structure respectively, separated with commas. It has only input parameters and thay can be seen in table C.12.

Function name	INPUT	OUTPUT
store_frame_struct	-	frames - cell structure with frames file - name of the file to write

Table C.12. `store_frame_struct`

Function `[subframe_struct] = generate_subframes(filename,nr_PGinSF,nr_SFinFR)`

This function reads the pages from the frame structure and makes subframes out of it according to [20]. Only the pages with correct crc check (equal to 'Y') are taken into account. Page number and navigation data are then assigned to a new structure, which is stored in cell array that has a size of one subframe. Then all subframes from one frame are assigned to a frame cell, which has the size of one frame. Then again all frames from one channel are stored in another cell array. Hence 3 cell arrays are allocated within this function. It is worth mentioning that not the whole subframe and/or frame structure will be completely filled with pages. It depends on the length of the data record from the receiver file. The structures are filled with the existing pages as they appear in the file according to their page numbers. Input/output parameters are given in table C.13.

Function name	INPUT	OUTPUT
generate_subframes	filename - name of the file to read nr_PGinSF - nr of pages in one subframe nr_SFinFR - nr of subframes in one frame	subframe_struct - extracted symbol sequence

Table C.13. `generate_subframes`

Function `store_subframes(subframe_struct,filename,pathname)`

This function is used to write subframes into separate ASCII files. Each file corresponds to one subframe.Firstly the frames are extracted out of the frame structure, then the subframes. Page number and navigation data are written into the file in each row. Format inside of the file is *page number*, *page data*. Input/output arguments are given in table C.14.

Function name	INPUT	OUTPUT
store_subframes	subframe_struct - subframe structure filename - name of the file to write pathname - name of the full path of the file	-

Table C.14. `store_subframes`

Flow chart of the main program where the functions are executed, **`main_getr.m`**, is presented in figure C.4.

GETR Symbols are first extracted from the file, and stored into a long vector of symbols. Then the first synchron pattern is found in that vector of symbols. With the index of the synchron pattern, one loop is performed over the sequence. It takes one page in each iteration, extracts the symbols from it, deinterleaves and decodes them, after which the crc check is performed. Those pages are written in the frame structure and stored in files. Each file contains frames from only one channel. From each frame the subframes are extracted and written into ASCII files.

The second part of this program implements some additional functions that are executed within the M-files in a form of **`run_rinex.m`** and **`run_yuma.m`**. First one writes navigation message parameters, such as Ephemeris, Clock corrections, GST to UTC corrections, Ionospheric corrections in Rinex files (see [26]), and the second one writes Almanac parameters to Yuma files.

Following functions need to be executed in order to export those files.

Function `[page] = page_struct_fnav(page_data,page_type,page_nr,plane_nr)`

This function is used to make a structure with navigation parameters obtained pages for the message type **FNAV** and the signal **E5a**. Structure of each page can be taken from [20] and input/output arguments are given in table C.15.

Function name	INPUT	OUTPUT
page_struct_fnav	page_data - navigation data page_type - type of page page_nr - page number plane_nr - orbital plane number	page - structure with nav parameters

Table C.15. `page_struct_fnav`

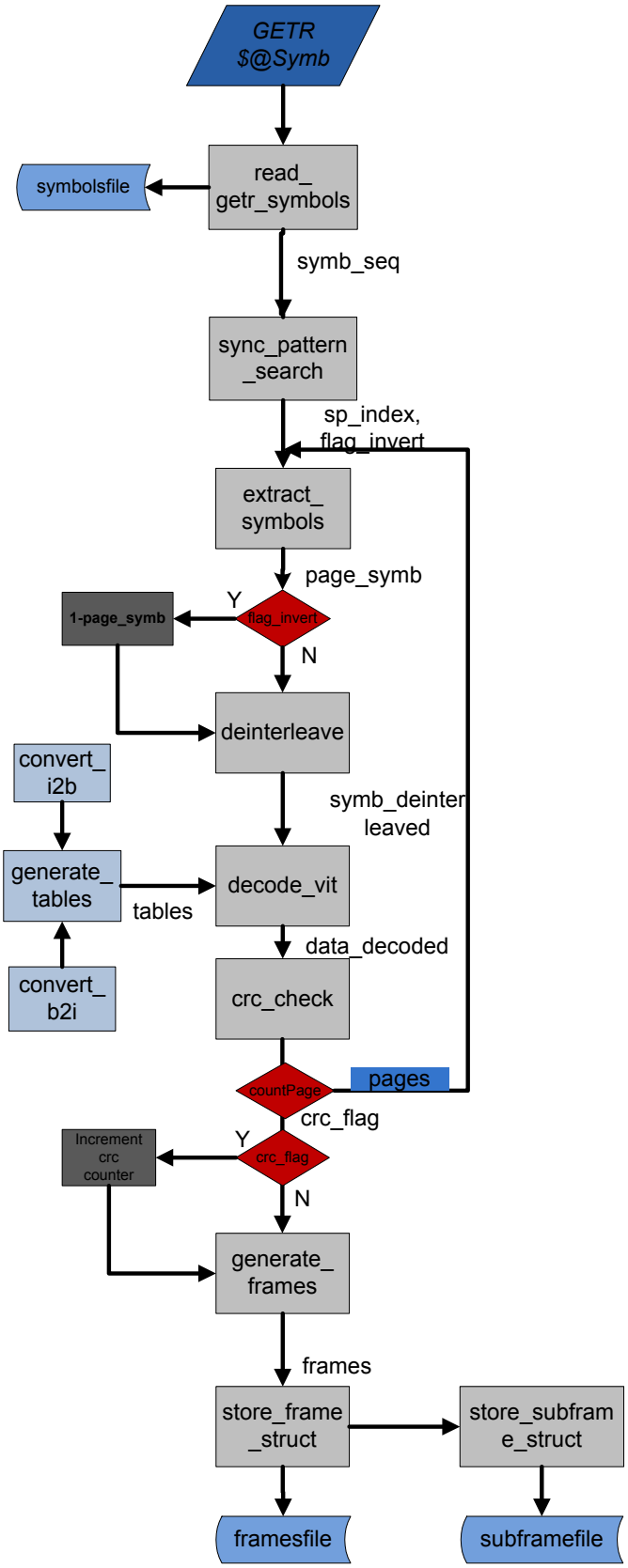

Figure C.4. Flow chart of the main program

Function `[page] = page_struct_inav(page_data,page_type,page_nr,packet_nr,SF_nr,plane_nr,k,msg_type)`

This function is used to make a structure with navigation parameters obtained from pages for the message type **INAV** and both signals, **E5b** and **E1B**. Structure of each page can be taken from [20] and input/output arguments are given in table C.16.

Function name	INPUT	OUTPUT
page_struct_inav	page_data - navigation data page_type - type of page page_nr - page number plane_nr - orbital plane number packet_nr - nr of packet with data SF_nr - subframe number k - almanac slot msg_type - message type	page - structure with nav parameters

Table C.16. `page_struct_inav`

Function `[int_value] = calculate_param(bin_num,flag)`

This functions computes the values of navigation parameters, that are given in binary form, and can be in two's complement notation. Input argument flag is set to 1 when two's complement computation has to be executed. MSB value (0 or 1) determines the sign for this operation. If number is not in two's complement notation it is simply converted to its integer value. Input/output parameters are given in table C.17.

Function name	INPUT	OUTPUT
calculate_param	bin_num - binary vector with nav data param flag - indicator of two's compliment	int_value - integer value of param

Table C.17. `calculate_param`

Before going into explanation of **`run_rinex.m`** and **`run_yuma.m`**, one important issue has to be discussed. According to GIOVE A+B SIS ICD [20] one Navigation Message Page is generated and has the format as in table C.18.

Sync	Res-1	PGCNT	SNF	NAVDATA	Res-2	CRC	Tail

Table C.18. GIOVE Page Layout

Theoretically only the NAVDATA field is needed for evaluation of navigation parameters. However, one of parameters, namely the System Issue of Data (SIOD) is calculated employing the SNF field as well as the NAVDATA field. From this reason, the pages stored in the frame and subframe structures contain data from NAVDATA field expanded by the SNF field, which is appended at the beginning, hence their length will be the length of SNF plus the length of NAVDATA field. This has to be considered when evaluating pages from the subframe structure.

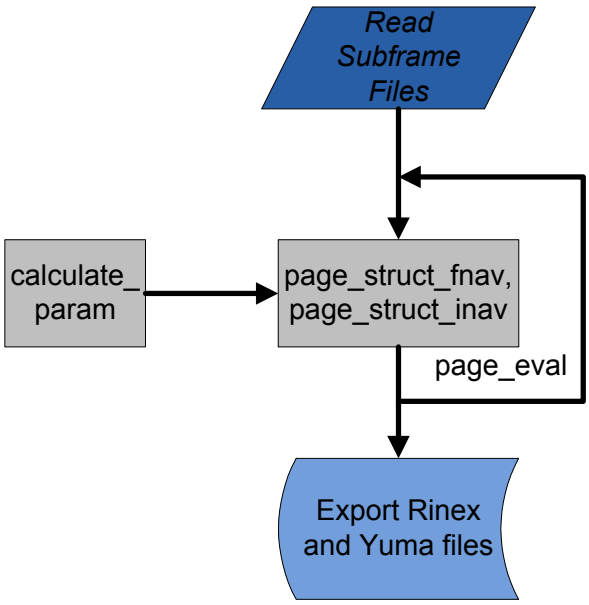

Figure C.5. Flow chart of Rinex and Yuma

C.2 Rinex and Yuma programs

RINEX: There are three main programs to run, all of them produce Rinex files, but for different messages. These programs are: **`run_rinex_E5a`**, **`run_rinex_E5b`**, **`run_rinex_E1B`**. For details about Rinex format, see [26]. **YUMA:** The same discussion stays for the Yuma format and the details can be taken from [5].

Program `run_rinex_E5a`

This program is used to evaluate one subframe from **F/NAV** message, signal **E5a** and write the extracted values in Rinex files. Pages from which navigation parameters are extracted, are the first four pages in each subframe. They are evaluated with execution of the function **`page_struct_fnav`**. Parameters are exported according to Rinex file description. Since most of the values have exponential format, which is in Matlab under Windows equal to 'E+000' or 'E-000', hence with three digits, and the original file provided in Rinex format under Unix has the notation 'E+00' or 'E-00', the conversion has to be made. It is done with string replace function, where the exponential value is first converted to a string and then replace with another string.

Program `run_rinex_E5b`

This program is used for the evaluation of two consequent subframes from **I/NAV** message, signal **E5b** and for writing the extracted values in Rinex files. Pages from which navigation parameters are extracted, are taken for both subframes. They are evaluated with execution of the function **`page_struct_inav`**. The same conversion is made here as in **`run_rinex_E5a`**.

Program `run_rinex_E1B`

This program is used for the evaluation of two consequent subframes from **I/NAV** message, signal **E1B** and for writing the extracted values in Rinex files. Pages from which navigation parameters are extracted, are taken for both subframes. They are evaluated with execution of the function **`page_struct_inav`**. The same conversion is made here as in **`run_rinex_E5a`**.

Programs `run_yuma_E5a`, `run_yuma_E5b` and `run_yuma_E1B`

These programs are used to produce files in Yuma format, for different navigation messages. The files contain Almanac parameters and those parameters reside in all pages of one frame. Hence a long data record with all full subframes within a frame is needed to extract those parameters. Programs are divided into segments using the cell mode and can be executed independently in most of the cases.

A small flow chart with execution of the used functions for Rinex and Yuma files is depicted in figure C.5.

D Noise in the channel

For the purpose of navigation channel simulation and as usual in transmission channel simulations, the model applied on the channel is the Additive White Gaussian Noise (AWGN) model. Its main characteristic is a white noise signal and it will be explained in the following section.

D.1 White Noise

White noise is a noise signal that contains all frequencies at the same time, similar as the white light contains all the colors of the spectrum.

In technical applications it is a random signal that has a flat power spectral density function (PSD), which is independent and constant over all frequencies. That implies an infinite power spectrum (integral of PSD over all frequencies), which is in praxis impossible to generate. However, some approximation can be made over a limited bandwidth where the PSD function still remains constant.

White noise model is used to describe a lot of real world processes, which are then modeled with some mathematical system model. If white noise is considered to be normally or Gaussian distributed (one important and often employed probability distribution) it is called the Gaussian noise. This kind of noise is used in communication system models to simulate the channel behavior and this kind of channel is named Additive White Gaussian Noise channel *AWGN*.

One of the important properties of White Gaussian Noise (WGN) is that its values are statistically independent. It means that the existence of one value does not have any influence on the existence of another value. From a statistical point of view, the conditional probability of one event, given the other one is equal to unconditional probability of the first event as the second event was not taken into account. Since the independence of two variables implies that they are also uncorrelated (but not vice versa!), it can be established that the white Gaussian noise is statistically uncorrelated. This holds for a case of autocorrelation of the white noise in time that gives a degree to which two time samples of the same random process are correlated. Autocorrelation function of a random process $x(t)$ can be defined as expected value of the product of two different time samples of this random process:

$$R_x(t,t+\tau) = E[x(t)x(t+\tau)] \tag{D.1}$$

In an AWGN channel model, uncorrelated white noise with a constant spectral density and Gaussian distributed amplitude is added to the channel. Due to the central limit theorem which emphasizes that the sum of a large number of statistically independent random variables (e.g. white noise) will also have a normal (Gaussian) distribution, it can be used as a very good approximation in the description of a channel.

Any random process *n(t)* is considered to be a white Gaussian noise process if it meets the following conditions:

1. $E[n(t)] = \mu = 0$ zero mean value
2. At every distinct time instant $t_1, t_2, \ldots, t_n$ the $n(t_1), n(t_2), \ldots, n(t_n)$ are the independent Gaussian variables

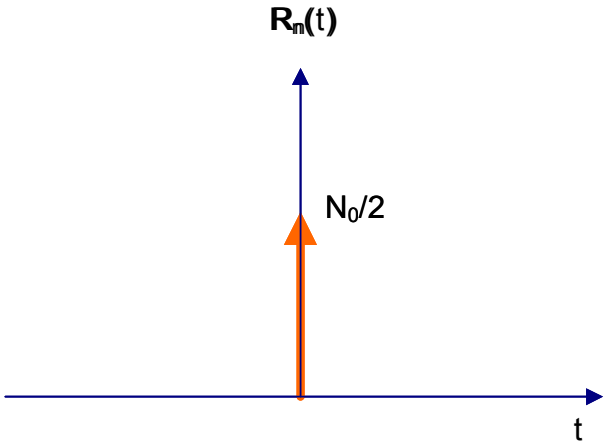

Figure D.1. Autocorrelation of white noise

The autocorrelation function of *n(t)* is a delta-distribution at the zero point, with an amplitude of $N_0/2$ and can be seen in figure D.1.

$$R_n = \frac{N_0}{2} \cdot \delta(\tau) \; for \; -\infty \leq \tau \leq \infty \tag{D.2}$$

N_0 is the power density of the noise in general; it is a noise power per unit of bandwidth, given by a formula in equation D.3:

$$N_0 = k \cdot T \tag{D.3}$$

Whereby k is the Boltzmann constant and T is the system noise temperature.

Since the PSD is calculated as the Fourier transformation of the auto-correlation function, it follows that the PSD of white noise is equal to $N_0/2$ - which coincides with the previous assumption about the constant (flat) spectral density of white noise processes. It is shown in figure D.2.

$$S_n(f) = \frac{N_0}{2} \; for \; -\infty \leq \tau \leq \infty \tag{D.4}$$

The average power has an infinite value since its bandwidth is infinite.

$$P_n = E[n^2(t)] = \infty \tag{D.5}$$

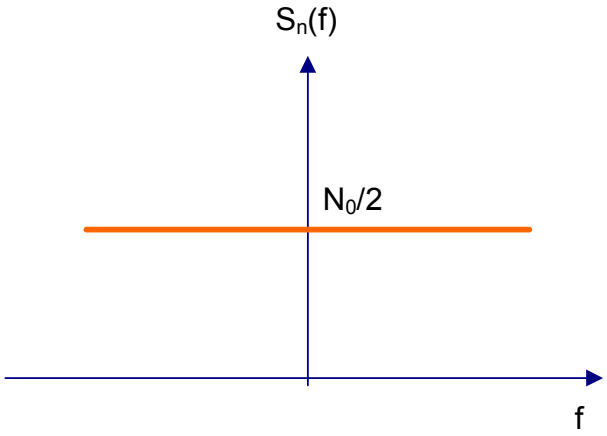

Figure D.2. Power spectral density of white noise

In a sampled signal, where the period of sampling is equal to T_s and the corresponding sample frequency is f_s, the power or variance of the WGN is given as in equation D.6:

$$\sigma^2 = \frac{PSD}{T_s} = PSD \cdot f_s \tag{D.6}$$

It is further equal to:

$$\sigma^2 = \frac{N_0}{2 \cdot T_s} \; since \; PSD = \frac{N_0}{2} \tag{D.7}$$

In lot of practical channel simulations, in order to view the performance of the channel, the Energy per bit to Noise power spectral density ratio (Eb/N_0) has been used. It is a parameter known as "normalized version of signal-to-noise ratio (SNR)" whereby E_b is the energy of the bit calculated from the power of each sampled bit $E_b = P_s \cdot T_s$. Combining this expression with the expression for the Power density of the noise, obtained from the Equation D.7 as: $N_0 = 2 \cdot T_s \cdot \sigma^2$ one can obtain the relation for $E_b N_0$:

$$\frac{E_b}{N_0} = \frac{P_s \cdot T_s}{2 \cdot T_s \cdot \sigma^2} = \frac{P_s}{2\sigma^2} \tag{D.8}$$

The power of a sampled signal is equal to a square of its amplitude, which is usually normalized to one, i.e. $P_s = 1$. Hence, one can obtain an expression for a parameter known as standard deviation as:

$$\sigma = \frac{1}{\sqrt{2\frac{E_b}{N_0}}} \tag{D.9}$$

This expression can be used in E_b/N_0 calculations and simulations of the AWGN channel. In a practical applications, the best way to generate noise is to:

- generate random variable that is normally (Gaussian) distributed, with the mean value equal to zero and the standard deviation equal to one, and multiply it with obtained value for standard deviation.
- add it to the signal which is affected by the noise, and transmit that signal through the channel.

D.2 Error Probability

In communication systems noise can occur in various forms. Much of those can be reduced, except the one that always follows electrical transmissions and cannot be eliminated - it is the thermal noise. Thermal noise is caused by the thermal movements of electrons.

It can be modeled as a white Gaussian noise, i.e. its amplitude will have normal or Gaussian distribution. This distribution describes the data as a cloud concentrated around one value, the mean value. It has probability density function (PDF) in characteristic bell form, where its maximum is its mean value at the same time. In the case of thermal noise, since it is modeled as white noise, the mean value will be zero - and that is where the name zero-mean Gaussian random process come from.

Power spectrum of thermal noise has its maximum at zero frequency, and goes to infinity slowly converging towards zero. Hence, white noise represents a good approximation of it.

If $n(t)$ is a random function, its Gaussian PDF is characterized by the expression:

$$pdf = p(n) = \frac{1}{\sigma\sqrt{2\pi}} \cdot exp\left(-\frac{1}{2}\left(\frac{n}{\sigma}\right)^2\right) \tag{D.10}$$

Here σ^2 is the variance of n. Standard Gaussian distribution has variance equal to one, $\sigma^2 = 1$, hence its PDF will be:

$$pdf = p(n) = \frac{1}{\sqrt{2\pi}} \cdot exp\left(-\frac{n^2}{2}\right) \tag{D.11}$$

Graphical presentation of pdf function is given in figure D.3.

A random signal is often represented as the sum of a Gaussian noise random variable n and a dc signal a.

$$z = a + n \tag{D.12}$$

Inserted in equation D.10 the PDF will be

$$pdf = p(z) = \frac{1}{\sigma\sqrt{2\pi}} \cdot exp\left(-\frac{1}{2}\left(\frac{z-a}{\sigma}\right)^2\right) \tag{D.13}$$

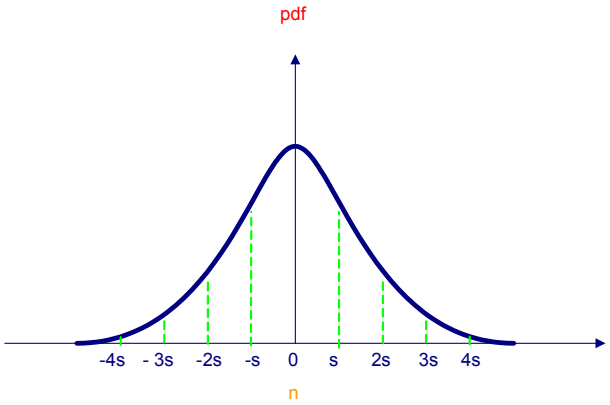

Figure D.3. Gaussian probability density function

After the transmitted signal passed various phases in its way to receiver, it finally comes to demodulator/detector. In a process of signal demodulation and detection the main task of the demodulator/detector is to recover the transmitted signal making the right decision based on probability criterion described bellow.

When receiving a signal that is disturbed by an additive white noise, it can be written in a form

$$r(t) = s(t) + n(t) \tag{D.14}$$

Where $r(t)$ stands for the received signal, $s(t)$ for the transmitted signal and $n(t)$ for the noise. When such a signal reaches the demodulator/detector, the correct decision about the original transmitted signal should be made. This process is usually implemented in two steps:

1. The received waveform is sampled to symbols with duration T_s

$$r(T) = a_j(T) + n_0(T) \quad j = 1,2 \tag{D.15}$$

Here is $a(t)$ desired signal component, and $n_0(T)$ white noise as described above. Since it is a Gaussian random variable with a zero mean, it implies that $r(T)$ is also Gaussian random variable with mean around a_1 or a_2 depending whether binary one or zero was transmitted.

Considering the probability density function of a Gaussian random variable n_0 as defined in D.10, the conditional probability density functions can be expressed as:

$$pdf(r|s_1) = \frac{1}{\sigma_0\sqrt{2\pi}} \cdot exp\left(-\frac{1}{2}\left(\frac{z-a_1}{\sigma_0}\right)^2\right) \tag{D.16}$$

for the first probable transmitted signal

$$pdf(r|s_2) = \frac{1}{\sigma_0\sqrt{2\pi}} \cdot exp\left(-\frac{1}{2}\left(\frac{z-a_2}{\sigma_0}\right)^2\right) \tag{D.17}$$

for the second probable transmitted signal.

These two PDFs are called likelihood functions and refer to the probability that the symbol that was transmitted through the channel was s_1 or s_2 having the received symbol r. One example of likelihood functions is presented in figure D.4.

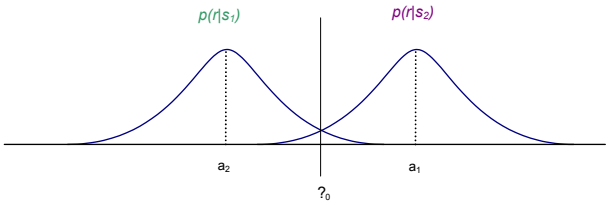

Figure D.4. Likelihood functions

Hereby, $p(r|s_1)$ is the likelihood function of s_1 and $p(r|s_2)$ likelihood of s_2, and γ_0 is a threshold value.

2. In a second step the measurement will be made; whether the symbol energy is greater or smaller than a threshold value. According to that value the appropriate decision for a symbol s_1 or s_2 will be made. One way to choose the threshold level is to minimize the error probability. For the case that the symbols s_1 and s_2 have the same likelihood probabilities, the optimum threshold value is represented by $\frac{a_1+a_2}{2}$ and passes exactly through the intersection line of the two likelihood functions. The decision made on this way corresponds to the signal with maximum likelihood function. This kind of detector is called maximum likelihood detector.

When an error occurs e.g. if s_1 is sent, but because of the noise the received signal $r(T)$ is less then a threshold γ_0 , it will result in a probability of error to be equal to the area left from the γ_0 and bellow the likelihood function of s_1. It can be expressed by cumulative conditional probability (cumulative because of the integral):

$$P(e|s_1) = \int_{-\infty}^{\gamma_0} p(z|s_1)dz \tag{D.18}$$

The same discussion holds for a case that signal s_2 is sent, and the received signal is greater than a threshold γ_0.

$$P(e|s_2) = \int_{\gamma_0}^{-\infty} p(z|s_2)dz \tag{D.19}$$

In case of a binary detection and according to the probability theory, the resulting probability of bit error will be:

$$P_B = P(e|s_1)P(s_1) + P(e|s_2)P(s_2) \tag{D.20}$$

As observed, a priori probabilities are equal $P(s_1) = P(s_2) = \frac{1}{2}$ and due to the symmetry of the PDFs:

$$P_B = \frac{1}{2}P(e|s_1) + \frac{1}{2}P(e|s_2) = P(e|s_1) = P(e|s_2) \tag{D.21}$$

It leads to a conclusion that probability of bit error is equal to the shadowed area under the left or right tail part of the likelihood functions. Integral over likelihood functions with appropriate limit values will have the form:

$$P_B = \int_{\gamma_0}^{\infty} \frac{1}{\sigma_0\sqrt{2\pi}} \cdot exp\left(-\frac{1}{2}\left(\frac{z-a_2}{\sigma_0}\right)^2\right) dz \tag{D.22}$$

Considering that $\gamma_0 = \frac{a_1+a_2}{2}$

Having $u = (z-a_2)/\sigma_0$ follows that $\sigma_0 du = dz$ and

$$P_B = \int_{\frac{a_1-a_2}{2\sigma_0}}^{\infty} \frac{1}{\sqrt{2\pi}} \cdot exp\left(-\frac{u^2}{2}\right) du \tag{D.23}$$

which is further equal to $Q - function$ in a form $Q(\frac{a_1-a_2}{2\sigma_0})$.

In mathematics, $Q - function$ is denoted as a special function (non elementary) whose values are listed in tabular form.

$$Q(x) = \int_{x}^{\infty} \frac{1}{\sqrt{2\pi}} \cdot exp\left(-\frac{u^2}{2}\right) du \tag{D.24}$$

Knowing this, the expression in D.23 gives a probability of a bit error as:

$$P_B = Q\left(\frac{a_1-a_2}{2\sigma_0}\right) \tag{D.25}$$

This probability should be minimized in order to obtain an optimal detector decision. In the AWGN channel it could be achieved with the filter that maximizes the argument of $Q - function$. Hence, minimizing the error probability P_B equals to maximizing $\frac{(a_1-a_2)}{2\sigma_0}$. When maximizing this, the maximum distance between a_1 and a_2 will be obtained (maxima of the likelihood functions), with $(a_1 - a_2)$ as a difference of the desired signal components at the time instant $t = T$.

For this case, the filter that is applied and that achieves the best performance is called matched filter. It delivers the maximum possible output SNR equal to $\frac{2E}{N_0}$ (see Appendix), where $\frac{N_0}{2}$ is a power spectral density of the white noise as demonstrated in Section 1.1.

Since the filter is matched to the input difference signal $[s_1(t) - s_2(t)]$ the output SNR at time $t = T$ can be written as:

$$\frac{S}{N} = \frac{(a_1 - a_2)^2}{\sigma_0^2} = \frac{2E_d}{N_0} \tag{D.26}$$

Here is the $(a_1 - a_2)^2$ equal to instantaneous power of the difference signal, and the variance σ_0^2 corresponds to the power of noise.

Combining this equation with the Equation D.25 yields to important result:

$$P_B = Q\left(\sqrt{\frac{E_d}{2N_0}}\right) \tag{D.27}$$

1. Case of unipolar signaling Here are the signals defined as following:

$s_1(t) = A \;\; 0 \leq t \leq T$ for binary 1
$s_2(t) = 0 \;\; 0 \leq t \leq T$ for binary 0

With energy of distance $E_d = (A-0)^2 T = A^2 T$ and average bit energy of $E_b = \frac{A^2 T}{2}$ having two possibilities for each symbol, the error probability becomes:

$$P_B = Q\left(\sqrt{\frac{A^2 T}{2N_0}}\right) = Q\left(\sqrt{\frac{E_b}{N_0}}\right) \tag{D.28}$$

2. Case of bipolar signaling Here are the signals defined as following:

$s_1(t) = A \;\; 0 \leq t \leq T$ for binary 1
$s_2(t) = -A \;\; 0 \leq t \leq T$ for binary 0

With energy of distance $E_d = (A+A)^2 T = 4A^2 T$ and average bit energy of $E_b = \frac{2A^2 T}{2} = A^2 T$ having two possibilities for each symbol, the error probability becomes:

$$P_B = Q\left(\sqrt{\frac{2A^2 T}{N_0}}\right) = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \tag{D.29}$$

In technical simulations of the AWGN channels, the other special function if often used, called *complementary error function*. The relation between the $Q - function$ is defined as:

$$Q(x) = \frac{1}{2} erfc\left(\frac{x}{\sqrt{2}}\right) \tag{D.30}$$

For the both cases of signaling it can be calculated that

$$1.\ P_B = Q\left(\sqrt{\frac{E_b}{N_0}}\right) = \frac{1}{2}erfc\left(\frac{E_b}{2N_0}\right) \tag{D.31}$$

for unipolar signaling and

$$2.\ P_B = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) = \frac{1}{2}erfc\left(\frac{E_b}{N_0}\right) \tag{D.32}$$

for bipolar signaling.