Improved Turbo Codes for Data Transmission

by

Cagri C. Tanriover BSc, MSc

Submitted to the Higher Degrees Committee of Lancaster University
for the Degree of Doctor of Philosophy

Department of Communication Systems
February 2002

Table of Contents

List of Figures

List of Tables

Abstract

Since their invention in 1993, turbo codes have been the centre of attention in the coding community, as they achieved near Shannon performance for the first time in the five decade history of the information theory. Improving the frame and bit error performance of the turbo decoders using short information frame lengths, and reducing the decoding latency are the two main problems that need to be addressed to realise the legendary turbo codes in future digital communication devices. This thesis introduces applicable and novel turbo coding concepts, which contribute to this challenging objective.

Part of this thesis introduces the multifold turbo codes, which allow fast convergence in iterative decoders, and improve the frame error performance of turbo codes besides achieving low bit error rates, with short interleaving degrees. A subclass of multifold turbo codes, namely the two-fold turbo codes, have been assessed in terms of their error performance, and codeword weight distribution.

Also, a new approach to soft value processing based on the soft value magnitude and sign progress patterns of information symbols during iterative decoding, has been discussed. It is shown that various soft information derived metrics can be defined for controlling the decoder iterations and estimating the channel conditions. The effectiveness of the above processing techniques is presented through simulation results.

Furthermore, novel turbo-based hybrid automatic repeat request schemes, which utilise four retransmission modes with different throughput and frame error performance are explored, and analysed for various information frame lengths.

Two image transmission applications with equal and unequal error protection using turbo codes have also been developed and assessed in terms of their visual error performance.

It is hoped that the novelty of methods and results presented in this thesis will be useful to researchers who are particularly interested in the applicability of iterative coding.

Declaration

The work described in this thesis, which has been carried out at the Department of Communication Systems at Lancaster University between October 1998 and February 2002, is my own except where stated otherwise. No part of this thesis has been previously submitted to any academic institution for a higher degree, however, parts of the research have already appeared in the form of conference or journal publications, which are listed below.

- Tanriover C., Chippendale P., Honary B., "*Turbo Code Application to Image Transmission*", IEE Colloquium, Turbo Codes in Digital Broadcasting-Could it Double Capacity?, London, UK, November 1999, pp 17/1-17/6.
- Chippendale P., Tanriover C., Honary B., "*Enhanced Image Coding for Noisy Channels*", Proc. 7th IMA International Conference, Cirencester, UK, December 1999, pp 94-103.
- Chippendale P., Tanriover C., Honary B., "*Turbo Coded Image Transmission over Noisy Channels*", Fourth Volume on Communication Theory and Applications, Research Studies Press Ltd., 2000, UK, pp 277-286.
- Tanriover C., Honary B., "*An Enhanced Decision Stage for Turbo Codecs-Statistical Tracking System (STS)*", 1st Annual PostGraduate Symposium on the Convergence of Telecommunications. Networking and Broadcasting, Liverpool, UK, June 2000, pp 257-262.
- Tanriover C., "*Crossover Codeword Error Detection Algorithm*", 1st Student Conference on Communication Systems and Applications, Lancaster, UK, July 2000, pp 53-58.
- Coulton P., Tanriover C., Wright B., Honary B., "*Simple Hybrid Type II ARQ Technique Using Soft Output Information*", IEE. Electronics Letters, Vol.36, No.20, September 2000, pp.1716-1717.
- Tanriover C., Xu J., Lin S., Honary B., "*Multifold Turbo Codes*", Proc. IEEE. International Symposium on Information Theory, June 2001, p 145.
- Tanriover C., Xu J., Lin S., Honary B., "*Improving Turbo Code Error Performance by Multifold Coding*", scheduled to appear in IEEE Communications Letters, May 2002.
- Tanriover C., Honary B., "*Crossover Codeword Algorithm for Iteration Control*", submitted to IEEE Semiannual Vehicular Technology Conference, Vancouver, Canada, September 2002.
- Tanriover C., Honary B., "*Image Transmission Using Unequal Error Protection with Two-fold Turbo Codes*", submitted to IEEE International Telecommunications Symposium, Natal, Brazil, September 2002.

Acknowledgements

I wish to express my gratitude to my supervisor Prof. Bahram Honary for his encouragement and support in the course of this research. I would like to thank Tandberg television for their financial support, and the staff of the Department of Communication Systems for all the help they have provided during the past three years. I am also grateful to my colleagues Simon Hirst and Keith Barratt for the invaluable technical discussions, suggestions, and their friendship. Special thanks go to Dr. Paul Chippendale for the coffee table conversations, which were both enjoyable and technically inspiring, and most of all for being so supportive during tough times as a good friend.

Finally, my deepest feelings go to my parents, Ciler and Kemal Tanriover for all the years of caring, love and support, and my brother Cagdas Tanriover, who is an important part of my life, as always.

Chapter 1

Introduction

1. Introduction

Samuel Morse's invention of the telegraph in 1837, followed by the invention of telephone in 1876, and the first wireless voice and data transmission in 1901, have marked the beginning of the telecommunications age, which has transformed our world into a global village that we live in today. The more recent developments (see Table 1.1), such as the vacuum triode, the transistor, and the microprocessor, have given birth to the most powerful device of the twentieth century - the digital computer. Convergence of the telecommunications and the computer technology has catalysed the evolution of the practical communication systems, which have linked the entire globe as well as the outer space. Stimulation of the growth of the social and economical interaction all over the world has had an avalanche effect on the growth of the communications industry, which has no foreseeable end and our global village is expected to get even smaller.

Year	Event
1837	Invention of telegraphy by Samuel Morse
1864	Theoretical study of the existence of radio waves by James Maxwell
1876	Invention of telephone by Alexander Graham Bell
1887	Experimental verification of Maxwell's electromagnetic wave theory by Heinrich Hertz
1894	Demonstration of radio communication over a distance of 150 yards by Oliver Lodge
1901	Radio signal transmission over 1700 miles across the Atlantic by Guglielmo Marconi
1904	Invention of the vacuum-tube diode by John Fleming
1906	Invention of the vacuum-tube triode by Lee DeForest
1926	Demonstration of the mechanical television by John Logie Baird
1928	Demonstration of the first all-electronic television system by Philo Farnsworth
1933	Development of frequency modulation (FM) by Edwin Armstrong
1936	First television broadcast by the British Broadcasting Corporation
1937	Development of the pulse coded modulation (PCM) by Alec Reeves
1947	Development of the matched filter technique by D.O. North
1948	Invention of the transistor by Walter Brattain, John Bardeen and William Shockley
1954	Completion of the first transatlantic coaxial cable
1958	Production of the first silicon integrated circuit (IC) by Robert Noyce
1962	Launch of *Telstar 1* that relayed TV signals between the USA and Europe
1971	Development of the first single-chip microprocessor by Intel Corporation
1971	Development of the Advanced Research Project Agency Network (ARPANET)
1972	Development of the cellular telephone concept by Motorola
1977	Installation of the first 140 Mbit/s optical fibre system between Hitchin and Stevenage

1980	Invention of the compact disc (CD) by Philips and Sony
1986	Installation of the first international submarine optical fibre system between the UK and Belgium, which is 113 km long
1988	Installation of the first transoceanic optical fibre system (*also known as the transatlantic telephone cable*) between Europe and the USA, which is 9360 km long
1996	Launch of the first wireless local loop (WLL) in the UK at transmission speeds up to 128 kbit/s by Ionica
1998	Start of the first terrestrial digital television broadcast in the UK

Table 1.1 Historical development of telecommunications and related systems.

A modern radio communication system can be represented as in Figure 1.1. Note that the modulator and the demodulator blocks are essential, and synchronization of the transmitter and the receiver is also necessary, whereas the encoder and the decoder blocks are optional [Sklar, 88] . In such systems, there are two basic limitations on performance; the bandwidth and the noise [Miller, 96] .

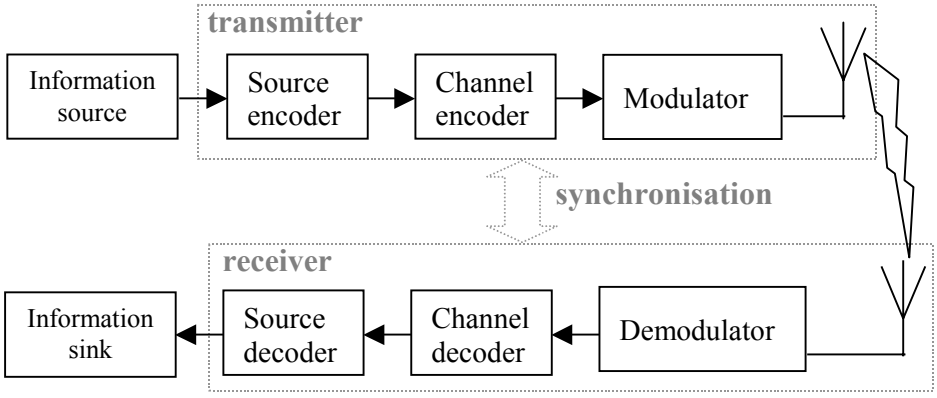


Figure 1.1 Basic components of a modern communication system

Bandwidth available in the radio frequency spectrum is limited (Table 1.2), and therefore needs to be utilized efficiently in a communication system. As we became increasingly dependent on communications, and the intense information traffic around the world started filling up the available frequency spectrum, bandwidth has become scarce and hence expensive. Today, bandwidth efficient digital modulation, compression and transmission techniques are still an active area of research, and the quest of transmitting the maximum amount of information in the minimum possible bandwidth continues.

Noise is the second limiting factor in all communication systems. During its journey through a communication system, the transmitted signal is distorted by noise from various sources such as the electronic components and the transmission medium. Unless precaution is taken, noise can introduce errors to the transmitted information to the extent that it is no

longer intelligible at the receiver side. In digital communication systems, information is channel coded so that such errors can be detected and corrected by using the added *redundancy*, i.e. the parity information.

Frequency	Designation
30-300 Hz	Extremely Low Frequency (ELF)
300-3000 Hz	Voice Frequency (VF)
3-30 kHz	Very Low Frequency (VLF)
30-300 kHz	Low Frequency (LF)
300-3000 kHz	Medium Frequency (MF)
3-30 MHz	High Frequency (HF)
30-300 MHz	Very High Frequency (VHF)
300-3000 MHz	Ultra High Frequency (UHF)
3-30 GHz	Super High Frequency (SHF)
30-300 GHz	Extra High Frequency (EHF)

Table 1.2 Radio Frequency Spectrum

As the amount of redundancy transmitted increases, the robustness of the information against noise increases since the erroneous information can be restored from a larger set of parity information. However, increasing the amount of redundancy to improve the reliability of transmission also increases the required channel bandwidth. In other words, there is a trade-off between maximizing the reliability of transmission and minimizing the necessary transmission bandwidth.

Shannon [Shannon, 48] was the first mathematician to successfully express the noise and the bandwidth constraints of a communication system in an elegant format in his paper using the channel capacity, whereby he marked the beginning of the information theory. Shannon mathematically proved that every communication channel could be characterized by the channel capacity, denoted C, such that information could be transmitted as reliably as required at any rate, R_t (in information bits/second), from one point to another, provided that $R_t<C$. He showed that for $R_t>C$ there was an unavoidable unreliability for information transmission. Shannon's work showed that in order to achieve efficient and reliable use of the transmission channel, coding was necessary [Massey, 92] . Shannon computed the capacity for the Gaussian channel as in (1.1), where B denotes the bandwidth and E_b and N_o denote the energy per bit and the one-sided power spectral density of the Gaussian noise, respectively. The maximum value for C can be calculated when B approaches infinity, which simplifies (1.1) to (1.2). When information bits are transmitted very close to the channel capacity, i.e. R_t is almost equal to the channel capacity, the minimum signal-to-noise ratio can be calculated as –1.59 dB, which is the well known Shannon limit for the additive white Gaussian channel.

$$C = B\log_2\left[1+\frac{E_b R_t}{BN_o}\right] \qquad \textbf{(1.1)}$$

$$\underset{B\to\infty}{C} = \frac{1}{\ln 2}\frac{E_b R_t}{N_o} \qquad \textbf{(1.2)}$$

After Shannon's paper, 50 years of research has worked towards developing coding schemes that could approach the Shannon limit and provide coding gains over the existing schemes. In the late 80s and the early 90s, practical coding schemes were approximately 2 dB away from the channel capacity, which was achieved by serial concatenated systems with extremely complex Viterbi decoders [Heegard & Wicker, 99] .

The invention of the turbo codes [Berrou et al, 93] in 1993, was the turning point in error coding, as for the first time in the history of the information theory, the achievable capacity came within our reach; the new class of codes were reported to operate only 0.7 dB away from the Shannon limit.

Turbo codes intrigued many researchers in the field of error control coding in the past few years, and significant improvements in capacity-approaching turbo codes were proposed afterwards. Frey and MacKay [Frey & MacKay, 00] came up with a novel turbo coding scheme, which was only 0.25 dB away from the Shannon limit. Recently, TenBrink [TenBrink, 00] showed that by using large block lengths and a high number of decoder iterations, turbo codes could operate 0.1 dB away from the theoretical limit.

In deep space applications, the coding gains achieved by the turbo codes over the existing schemes, is expected to have significant contributions to cutting down costs. Coding gain helps increase the distances over which reliable communication is possible, or it can help decrease the aerial size and the transmitter power, which in turn increases the battery lifetime of the satellite or a spacecraft. The scale of the economic incentive for approaching the channel capacity and achieving coding gains, justifies the relentless efforts in developing powerful channel coding techniques during the last five decades. In the mid-1960s, the value of coding gain on the downlink deep-space channel was estimated to be $1,000,000 per dB [Massey, 92] . The current value for the Deep Space Network is $80,000,000 per dB gain [Heegard & Wicker, 99] .

After an 8-year intense period of research on turbo codes, today the theory of these remarkable class of codes is well-understood, and it has become clear that by using very large information frame lengths (e.g. 1,000,000 bits/frame), and a high number of iterations (e.g. 300 iterations), the channel capacity can be reached with only 0.1 dB loss. However, the future commercial turbo coding applications such as GSM [Burkert et al, 96] , [Caire & Lechner, 96] , Universal Mobile Telecommunication System (UMTS) [Vivier, 01] and real-time voice and data services [Chan & Gerantiotis, 97] , turbo code frame lengths will need to be reduced (<5200 bits/frame) and the decoding iterations will need to be minimized (<10 iterations). Although practical limitations will force the turbo codes to operate below their actual potential, the future challenge will be to achieve the lowest bit and frame error rates with minimal system complexity and decoding delays.

Research documented in this thesis provides novel approaches to practical problems with turbo codes, and offers new methods of bit and frame error performance improvements, iteration control, and channel estimation related to turbo codes. Effort has been made to use simple codes and low-complexity decoding algorithms, keeping the applicability of the turbo codes in mind. The following is the roadmap for this thesis.

Chapter 2 discusses the channel noise, forward error correction and the automatic repeat request techniques, as a part of the theoretical background. Details of the Gaussian channel model, iterative encoding and decoding are presented in this chapter.

Chapter 3 introduces a novel turbo coding technique, namely the multifold turbo coding, which improves the error performance of the conventional turbo codes by bringing the codeword weight distribution closer to random. The structure of the generalized multifold turbo encoders and decoders are described in this chapter, and the two-fold turbo codes are analysed as an example in terms of their error performance and computational complexity. It is shown that significant coding gains over the conventional turbo coding schemes in terms of frame and bit error performance can be achieved by the two-fold codes besides saving up to 16 decoding iterations at the expense of 0.01% increase in decoder complexity.

An iteration control mechanism, namely the crossover codeword feedback system, which has low design and operational complexity, and utilises a user-defined BER threshold for terminating the decoder iterations, is introduced in Chapter 4. It is shown that the soft value sign change patterns during iterations - identified by the new crossover coding

algorithm - can be used to define a degree of reliability for each decoded information symbol. The chapter concludes with the error performance and the complexity analysis of the proposed iteration control system.

Chapter 5 presents an alternative channel estimation technique to the existing methods, which is suitable for AWGN channels. The new method provides sufficient estimation accuracy to achieve near-ideal BER performance at the expense of a small additional decoding delay. As it will be explained, the channel information calculated after decoding one information frame is used for decoding the following frame- a method that exploits the turbo decoder's low sensitivity to channel estimation errors. The chapter addresses the relationship between the soft extrinsic information and the channel noise, and describes how the soft value progress during iterations can be used to define a channel estimation metric. The error performance and the computational overhead evaluation of the new channel estimation technique conclude this chapter.

A new error detection algorithm and a hybrid ARQ scheme, that combines the two desirable features, namely the high detection accuracy and low processing complexity, are proposed in Chapter 6. In this chapter a soft-value derived detection technique and its incorporation into a turbo coded ARQ scheme with four retransmission strategies are explained in detail. The performance evaluation of the new scheme is presented in terms of error detection, information throughput and frame error rate. An analytical approach to the computational overhead of the proposed hybrid ARQ scheme is also included.

Chapter 7 constitutes two turbo coded image transmission applications. The first application is a serial concatenation of a turbo code with the compressed Absolute Picture Element (APEL) [Chippendale, 98] , [Chippendale et al, 99c] source coding developed at Lancaster University. In order to increase the error resilience of APEL image transmission at low signal-to-noise ratios, protection of the APEL-coded data via turbo codes has been investigated, and it has been compared to the turbo coded bitmap (BMP) and the Joint Photographic Experts Group (JPEG) [Wallace, 91] images. The second turbo code application in Chapter 7 is related to the transmission of the hue (H), saturation (S) and luminance (L) colour components of digital still images. Description of colour using hue, saturation and luminance has been introduced by the International Commission on Illumination (CIE) [CIE, 00] , in 1933, and relates very closely to human perception of colour. In this application, an unequal error control scheme, which is developed by the modification of the two-fold turbo code introduced in Chapter 3, developed for the reliable

transmission of HSL colour components over the AWGN channel is discussed. The proposed unequal error protection scheme is compared to an equivalent turbo coded equal error protection scheme, and the achieved error performance improvement is presented.

The final chapter of the thesis briefly summarises the significant results obtained in each chapter and it outlines further areas of research that may be viewed as extensions of the work presented in this document.

Chapter 2

Background Theory

2. Background Theory

This chapter is aimed at discussing the necessary theoretical concepts, upon which the research in the following chapters is based.

The three main sections in this chapter are, the channel noise, forward error correction, and the traditional automatic repeat request schemes.

The first section describes the general characteristics of channel noise with particular emphasis on the Gaussian noise. The second section is dedicated to the forward error correction. Revision of block and convolutional codes, as well as the iterative coding and decoding concepts are the primary points of focus in this section. The final section of the chapter is an overview of the automatic repeat request techniques, in which the traditional methods are explained.

2.1. Channel Noise

The term *noise* describes the unwanted waves that distort the message signals, which are transmitted in a communication system. The sources of noise can either be external or internal to the system. Atmospheric, galactic and man-made noises are classified as external, and the noise generated by the spontaneous fluctuation of voltage and current levels in electrical circuits is considered internal [Haykin, 94] .

Most types of noise are the results of a number of statistically independent and random events. Therefore, it is not possible to know the parameters of noise that deviates the received message signal from the transmitted signal, accurately. Due to its random nature, noise is characterized with the aid of probabilistic models. Using such models make the noise analysis possible since a probability density function (pdf), can be used to predict the unknown parameters of noise such as its magnitude [Cooper & McGillem, 88] . Depending on the statistical nature of channel noise, various pdfs can be used for channel modeling. Within the scope of this thesis, only one pdf, namely the Gaussian, will be discussed in detail. Further information on other pdfs can be found in [Taub & Schilling, 86] and [Chase & Bown, 86] .

The Gaussian (also called the *normal*) probability function is of great importance because many naturally occurring events are characterized by random variables with a Gaussian density. Another importance of the normal distribution is its involvement in the *central-*

limit theorem, which basically states that the probability density sum of *N* independent random variables tend to approach a normal density as the number *N* increases. Also according to this theorem, the mean and variance of this normal density are, respectively, the sum of the means and the sum of the variances of the *N* independent random variables [Taub & Schilling, 86] .

The probability density function of the normal distribution is described as in (2.1) and can be represented as shown in Figure 2.1, where the mean and the standard deviation of the distribution are denoted as μ and σ, respectively.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left[\frac{(x-\mu)}{\sigma}\right]^2} \qquad \textbf{(2.1)}$$

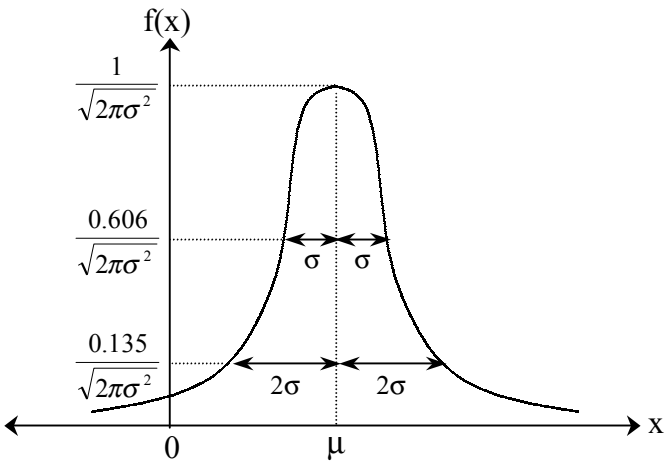


Figure 2.1 Probability density function of the normal distribution

Note that the total area under the pdf curve is equal to the probability of the continuous random variable *x* to vary between positive and negative infinity, which is equal to 1. As the variable *x* is continuous, the aforementioned area is calculated by integrating *f(x)* within the +∞ and -∞ range, as indicated in (2.2).

$$\int_{-\infty}^{+\infty} f(x)dx = 1 \qquad \textbf{(2.2)}$$

The integration of *f(x)* is difficult, which makes the probability calculations a complicated task. However, the integration of *f(x)* has been directly related to the *error function*, denoted *erf*(.), which has been tabulated and is readily available. The error function helps to calculate the probability of a random variable only in the positive range, as indicated in (2.3). As the normal pdf is symmetrical about the *y*-axis, the positive range limitation does not cause any problem; the obtained result also applies in the negative range.

$$erf(u) = \frac{2}{\sqrt{\pi}} \int_0^u e^{-u^2} du \qquad \textbf{(2.3)}$$

Also, the *complementary error function* (*erfc*(.)), is an alternative mathematical tool that can be used in the calculation of the probabilities mentioned above. The *erfc(.)* is related to the *erf(.)* as indicated in (2.4), and the range of the *erfc(.)* integral varies between a given positive random value and the positive infinity.

$$erfc(u) = 1 - erf(u) = \frac{2}{\sqrt{\pi}} \int_u^{+\infty} e^{-u^2} du \qquad \textbf{(2.4)}$$

Besides the pdf of the channel noise, its *colour* is another important parameter that needs to be taken into account in communication systems. The random noise types encountered in most communication channels are considered to be normally distributed and white. Just like the colour white constitutes all colours in the visible spectrum, the white noise pdf is flat over the entire frequency range of the communication channel. In most communication systems, noise is assumed to be effective within a finite bandwidth, due to the filtering in the communication receivers. Let us now consider how the random noise is associated with the band-limited white noise, and what the term channel noise will refer to within the scope of this thesis.

In order to determine if there is any correlation between the samples of a noise voltage recorded over a long period of time, T, we would compare two neighbouring samples at a time that are separated by a small length of time, τ, as expressed in (2.5). The expression in (2.5) is called the *time-averaged autocorrelation function* of the noise voltage *v(t)*.

$$R(\tau) = \lim_{T \to \infty} \frac{1}{T} \int_0^T v(t)v(t+\tau)dt \qquad \textbf{(2.5)}$$

Considering that the noise voltage consists of a large number of statistically independent samples, which means that $v(t+\tau)$ does not depend on $v(t)$, unless τ=0, the expression in (2.5) can take two values, as indicated in (2.6).

When τ is equal to zero, $R(\tau)$ is equal to $N_o/2$, where N_o denotes the power spectral density in watts/Hz. For nonzero τ values, the product $v(t)v(t+\tau)$ can either be negative or positive with equal likelihood. Therefore, if the average of the noise samples is zero, $R(\tau)$ is also equal to zero for $\tau \neq 0$.

$$R(\tau)=\begin{cases}(N_o/2)\delta(\tau) & \tau=0\\ 0 & \tau\neq 0\end{cases} \qquad \textbf{(2.6)}$$

In time domain, the $R(\tau)$ can be represented in terms of the Dirac function, $\delta(\tau)$. The Fourier transform of the $R(\tau)$ gives the spectral power density function, denoted $S(f)$. The graphical representations of the two functions are shown in Figure 2.2. Note that the power profiles of the noise samples throughout the entire frequency range are equal to $N_o/2$, which indicates that the white noise has infinite average power.

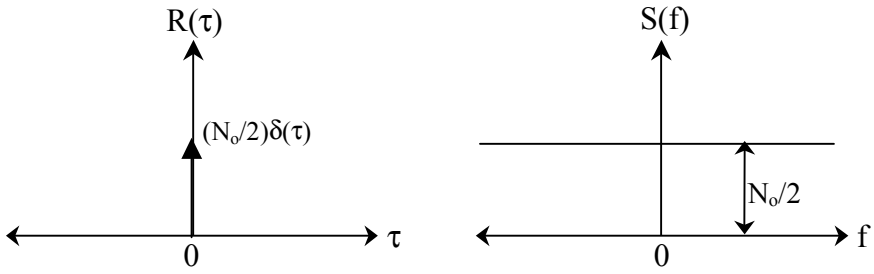


Figure 2.2 Autocorrelation, $R(\tau)$, and spectral power density, S(f), functions of white noise with infinite bandwidth

Considering the characteristics of the white noise, it can be stated that it is an idealised process, which cannot be realised in practical communication systems. It is important that the white noise only exists at the input of a communication receiver. As noise passes through the system, it is filtered and therefore is band-limited. If the bandwidth of the receiver is B, then the ideal representation in Figure 2.2, can be reconstructed as in Figure 2.3, for the double sided spectrum.

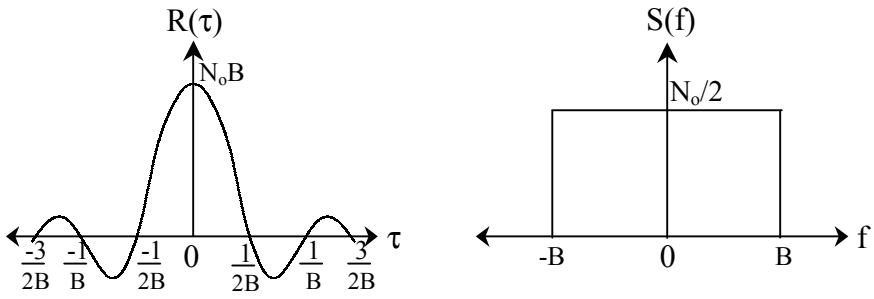


Figure 2.3 Autocorrelation, $R(\tau)$, and spectral power density, S(f), functions of white noise with limited bandwidth

In this case, the noise power, P, that is taken into account in practice is equal to the area under the $S(f)$ curve in Figure 2.3, which is expressed in (2.7). As it can be seen, the noise power of interest is the portion that falls within the pass band of the filter at the receiver, and has finite magnitude.

$$P=N_oB \qquad \textbf{(2.7)}$$

In the rest of this thesis, the channel noise will refer to the band-limited additive white Gaussian noise (AWGN), unless stated otherwise.

2.2. Forward Error Correction

The forward error correction (FEC) schemes are classified as either block or convolutional codes. In this section, these two basic classes of codes are visited, and special emphasis is placed on the convolutional codes, due to their particular relevance to this research.

2.2.1. Linear Block Codes

When two codewords are added in modulo-2 arithmetic, and their sum is equal to a codeword that is also a member of the same codeword space, such codes are said to be *linear*.

A block encoder accepts *k* message bits at a time and generates (*n*-*k*) redundant bits (or parity bits), which are computed according to a prescribed mathematical rule that defines the structure of the code. Such block codes are called *(n,k) linear block codes*.

If we have a block of *k* message bits, the codeword space constitutes 2^k different codewords. The (*n*-*k*) parity bits are the modulo-2 sums of the *k* message bits.
In order to explain the block coding, let us assume that a 1-by-*k* message vector **a**, 1-by (*n*-*k*) parity vector **b**, and a 1-by-*n* code vector **c** are defined as follows:

$$\mathbf{a}=[a_1, a_2, \ldots\ldots\ldots, a_k]$$
$$\mathbf{b}=[b_1, b_2, \ldots\ldots\ldots, b_{n-k}]$$
$$\mathbf{c}=[c_1, c_2, \ldots\ldots\ldots, c_n]$$

Then, the parity bits can be expressed as in (2.8), where **P** denotes the *k*-by-*(n-k)* coefficient matrix.

$$\mathbf{b}=\mathbf{aP} \qquad \textbf{(2.8)}$$

P can be expressed as,

$$\mathbf{P}=\begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1(n-k)} \\ p_{21} & p_{22} & \cdots & p_{2(n-k)} \\ \cdots & \cdots & \cdots & \\ p_{k1} & p_{k2} & \cdots & p_{k(n-k)} \end{bmatrix}$$

where p_{ij} can either be 1 or 0. The code vector **c** can be expressed as in (2.9), where **G** stands for the *k*-by-*n* generator matrix.

$$\mathbf{c=aG} \qquad \textbf{(2.9)}$$

The generator matrix consists of two parts; the *k*-by-*k* message identity matrix, $\mathbf{I_a}$, and the coefficient matrix, as shown in (2.10). One important feature of **G** is that its *k* rows are linearly independent, which means none of the rows can be expressed as a linear combination of the remaining rows.

$$\mathbf{G=[I_a|P]} \qquad \textbf{(2.10)}$$

An alternative way of expressing the relationship between the message bits and the parity bits of a linear block code is using the *(n-k)*-by-*n* parity check matrix, **H.** If the *(n-k)*-by-*(n-k)* parity check identity matrix is denoted $\mathbf{I_b}$, then **H** can be expressed as in (2.11), where $\mathbf{P^T}$ is the transpose of **P**.

$$\mathbf{H=[P^T|I_b]} \qquad \textbf{(2.11)}$$

The relationship (2.12) between the **H** and the **G** matrices, is significant from a decoding point of view, as it will be discussed shortly.

$$\mathbf{HG^T=GH^T}=0 \qquad \textbf{(2.12)}$$

If we multiply both sides of (2.9) by $\mathbf{H^T}$, the parity check equations in (2.13) equals to zero.

$$\mathbf{cH^T=aGH^T} \qquad \textbf{(2.13)}$$

Now let us consider sending **c** over a noisy channel, which is received as the noisy vector **r** as expressed in (2.14), where **e** denotes the error vector.

$$\mathbf{r=c+e} \qquad \textbf{(2.14)}$$

It should be noted that, for $i=1,2,...,n$, if $e_i=1$, an error in bit position *i* occurs. In other cases where $e_i=0$, bit *i* is received error free. The decoder needs to determine the vector **c** from the vector **r** without any a priori knowledge of **e**. Decoder can achieve this by calculating a *1*-by-*(n-k)* vector called the *syndrome*, **s**. The **s** vector is defined as in (2.15). Substituting (2.14) into (2.15), unveils the necessary error-correcting tool that the decoder needs, which is given by (2.16).

$$\mathbf{s=rH^T} \qquad \textbf{(2.15)}$$

$$\mathbf{s = (c+e)H^T = cH^T+eH^T = eH^T} \qquad \textbf{(2.16)}$$

As (2.16) shows, the **H** matrix allows us to compute **s**, which only depends on the error pattern **e**. This decoding technique is called *syndrome decoding*.

The number of errors that can be corrected by a linear block code depends on the *minimum Hamming distance (d_{min})* of the code, which is defined by the smallest Hamming weight of the nonzero codewords in the code. The Hamming weight of a codeword is defined as the number of nonzero bits in a codeword. A linear block code can correct up to t errors if (2.17) is satisfied. Note that $\lfloor x \rfloor$ denotes the largest integer that is less than or equal to x.

$$t \leq \left\lfloor \frac{d_{\min} - 1}{2} \right\rfloor \qquad \textbf{(2.17)}$$

For further information on block codes the reader is referred to [Lin & Costello, 83] , [Sklar, 88] , [Wicker, 95] , and [Drury et al., 01] .

2.2.2. Convolutional Codes

A convolutional encoder accepts the incoming information bits serially, rather than in blocks (as in block codes). It can be characterised by the *constraint length*, denoted K, which is defined as the number of positions that a single information bit can influence the output. In a convolutional encoder with an M-stage shift register, the memory is equal to M information bits and the K parameter is equal to M+1.

Figure 2.4 shows the convolutional encoder for the g(7,5) code, with the ex-or connection represented in octal form, for which M=2 and K=3. If the input information bit, u_k, appears with the generated parity bits p_{1k} and p_{2k} at the output, the code is said to be *systematic* and the overall code rate is equal to 1/3. When the codeword constitutes only the generated parity bits, the code becomes *non-systematic* and the rate increases to ½. Higher rate codes can be obtained by puncturing the output codeword, and transmitting the selected ones over the communication channel.

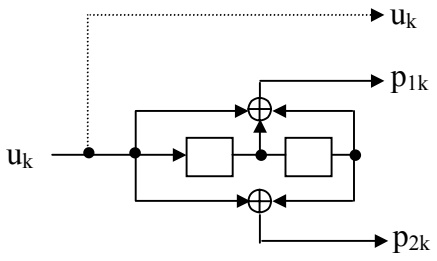


Figure 2.4 Encoder diagram for the g(7,5) convolutional code

Traditionally, the convolutional codes can be portrayed graphically by using one of the three equivalent diagrams: tree, trellis, and state diagram. In this section, the trellis representation will be presented, as it is more commonly used. More information on alternative representation methods can be found in [Sklar, 88] and [Carlson, 86] .

The trellis diagram for the non-systematic g(7,5) code is shown in Figure 2.5. Each state is labelled with a binary number that indicates the content of the shift registers at a given time. Each branch is also labelled with a binary number which shows the parity output of the encoder in response to an input information bit u_k. Dashed and solid branches correspond to u_k=1 and 0, respectively. The path starting from the state 00, and ending at a future state *s*, provides information on the trellis transitions, input information bit sequence as well as the generated codeword up to *s*. Notice that as the convolutional coding is continuous, the trellis does not terminate. Encoding continues until no more information bits are fed to the encoder.

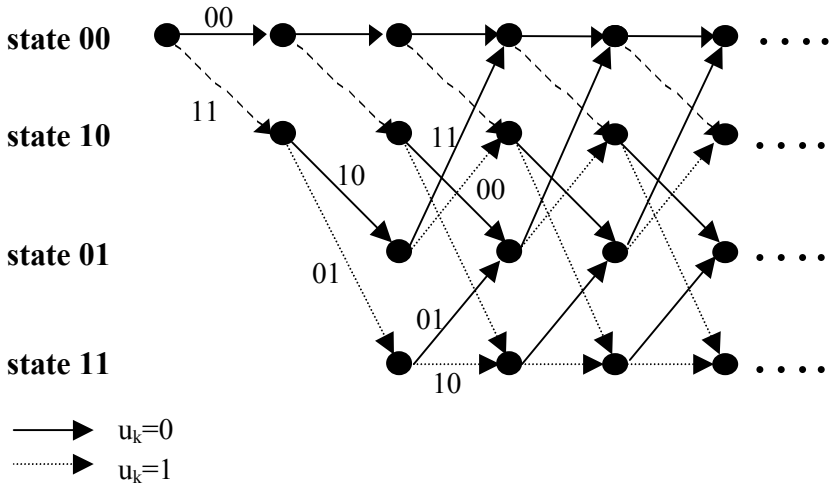


Figure 2.5 Trellis diagram for the non-systematic g(7,5) convolutional code

The convolutional codes can be decoded by using one of the three algorithms: maximum likelihood decoding (Viterbi algorithm), sequential decoding or majority logic decoding. As it is widely used, the Viterbi algorithm is going to be explained in detail with an example. Further details on the other two decoding methods are available in [Lin & Costello, 83] .

The Viterbi algorithm searches for a survivor path through the trellis, which minimizes the Hamming distance between a received vector and the valid codewords. While doing this, all the possible transitions in the trellis are taken into account, and the distance metrics are constantly updated. In order to fully understand how the algorithm works, let us consider a worked example.

Assume that an all-zero sequence that is encoded using the non-systematic g(7,5) code is sent over a noisy channel, and the transmitted parity sequence is received as r={0**1**000**1**0000...}. Notice that there are two errors, which are highlighted in bold. The decoding steps have been illustrated in Figure 2.6. At each step, the Hamming distance between the two bits of the received codeword and the branch values are calculated, and the cumulative distance metric is assigned to the future state. In the figure, the distance metrics are indicated at the rightmost section of the trellis. It should be noted that as long as the number of branch entries to a future state, *s*, is one, no path elimination is necessary, as the path of the entering branch will also be one of the survivor paths. This is true for the first two decoding steps in our example. In the third decoding step, for each future state, there are two paths that merge. The distance metrics for each candidate survivor path is indicated at the corresponding state in Figure 2.6. After the calculation of the distance metrics, at each state, the path with the highest metric is discarded, and hence one survivor path per future state is stored. If the metrics for both candidate paths were the same, then the survivor would have to be chosen randomly. At the end of decoding step 5, the final survivor is found to be the all-zero sequence, which is actually the transmitted sequence.

Generally, in Viterbi decoding the received information sequence is assumed to be infinite. Therefore, hard decision needs to be applied after a certain number of decoding steps in order to reduce the storage requirements in practice. As the number of decoding steps increases, after a point, all the candidate survivors stem from the same state, which means that none of the discarded paths can change the source path at all. Sklar [Sklar, 88] has suggested that after decoding $5 \cdot K \cdot (2^{K-1})$ trellis depths, all candidate survivor paths have the same source path, and therefore hard decision can be applied to the oldest depth in the trellis. Hence, the number of stored trellis depths is sufficient to achieve near-optimum decoder performance.

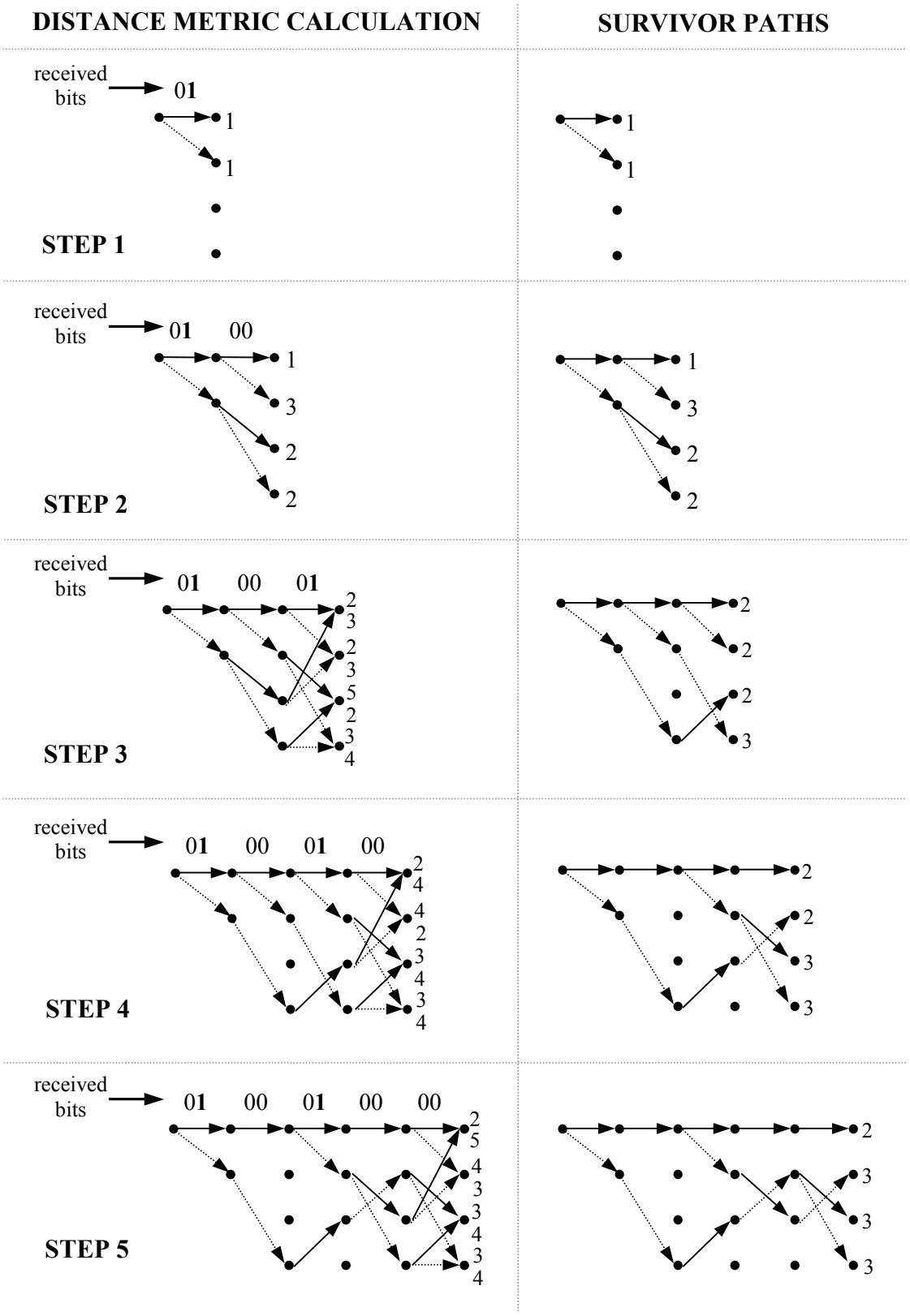


Figure 2.6 Viterbi decoding example

2.2.3. Iterative Coding Ingredients

This section deals with three key concepts in turbo decoding, which are interleaving, code concatenation, and soft/hard demodulators.

The concept of interleaving is explained in the first subsection with one example of algebraic interleaving. A brief revision of code concatenation follows the interleaving section. Comparison of soft and hard output demodulators is discussed in the final subsection. The soft output description is limited to the modulation scheme and the noise model used in this research, in order to give the reader a clear insight into the actual scope of the work.

2.2.3.1. Interleaving

Interleaving can be defined as the re-ordering of a set of elements that belong to a fixed alphabet. An interleaving function, f_π, takes the current position (x) of an element, z, in an array as its argument, and outputs its new position (y) in a new array. An interleaver is the device that performs this operation. De-interleaving function is the inverse of f_π, which maps z back to its original position. Figure 2.7 illustrates the principles of interleaving and de-interleaving. Assume that z_i denotes the element in the i^{th} position in an array. In this case the interleaver would move z from its current x position to a new y position, whereas the de-interleaver would perform the opposite.

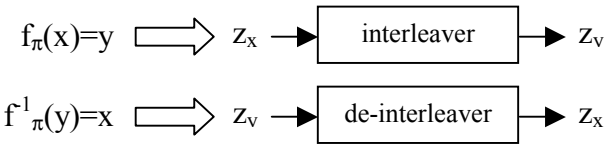


Figure 2.7 Interleaving and de-interleaving principle

Interleaving strategies can be categorised as block, convolutional, algebraic and pseudorandom depending on the f_π function used. In this section, only Berrou's algebraic interleaver [CCSDS, 98] will be explained, as it has been used throughout this thesis. For further information on interleaving, the reader is referred to [Heegard & Wicker, 99] , [Sklar, 88] and [Taub & Schilling, 86] .

The objective of Berrou's interleaving algorithm is to take k binary numbers in an array S, and to construct a re-ordered array $\Pi(S)$. Before explaining the interleaving algorithm, it should be pointed out that $\lfloor s \rfloor$ denotes the largest integer less than or equal to s, *mod* indicates modulo operator and p_q denotes one of the following eight prime integers:

p_1=31, p_2=37, p_3=43, p_4=47, p_5=53, p_6=59, p_7=61, p_8=67

Also, k can be expressed as the product of two positive integers, i.e. $k=k_1k_2$, where k_1 is an even integer.

The steps of the interleaving algorithm are as follows:

step 1 $m=(x-1) \bmod 2$

step 2 $i=\left\lfloor \frac{x-1}{2k_2} \right\rfloor$

step 3 $j=\left\lfloor \frac{x-1}{2} \right\rfloor - ik_2$

step 4 $t=(19i+1) \bmod \frac{k_1}{2}$

step 5 $q=t \bmod 8+1$

step 6 $c=(p_q j+21m) \bmod k_2$

step 7 $y=f_\pi(x)=2(t+c\frac{k_1}{2}+1)-m$

By using the above algorithm, the bit in position x in S is assigned a new location y in the $\Pi(S)$ array. Once the interleaving pattern is determined, the de-interleaving pattern can be stored as a look-up table.

As an example, if k_1 and k_2 are chosen as 4 and 5, respectively, the interleaving degree, k would be 20. Assume that we are trying to find the interleaved position, y, of the bit in position 7. Following the above seven steps, this can easily be determined:

step 1 $x=7 \Rightarrow$ **m=0**

step 2 $x=7, k_2=5 \Rightarrow$ **i=0**

step 3 $x=7, k_2=5, i=0 \Rightarrow$ **j=3**

step 4 $k_1=4, i=0 \Rightarrow$ **t=1**

step 5 $t=1 \Rightarrow$ **q=2**

step 6 $t=1, p_2=37, j=3, m=0, k_2=5 \Rightarrow$ **c=1**

step 7 $x=7, t=1, c=1, m=0, k_1=4 \Rightarrow$ **$y=f_\pi(7)=8$**

The de-interleaver look-up table for the above example would map the bit in the 8th position of the interleaved array back to the 7th position.

2.2.3.2. Concatenation of Component Codes

Concatenated coding systems use two or more component codes to enhance the error performance at a moderate complexity level. Codes can either be concatenated serially or in parallel.

In serial concatenation (Figure 2.8), the source information is first encoded by an outer code, followed by an inner code. The two component codes are separated by an interleaver. At the receiving end, the inner code is decoded before the outer code. The component codes operate jointly to combat bursty and random errors. As an example, the CCSDS standard uses a half rate K=7 convolutional inner code and an outer Reed-Solomon (RS) code to correct the bursty output of the convolutional decoder.

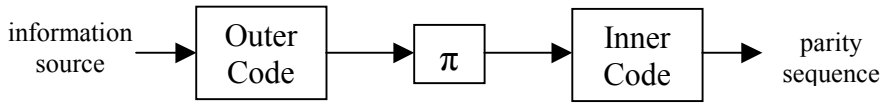


Figure 2.8 Serial concatenation

One advantage of the parallel concatenation over the serial one is that it provides a higher code rate than the serial concatenation when the code is not punctured. If the outer and the inner code rates in a serial concatenation scheme is given by R_1 and R_2, respectively, the overall code rate, R, can be expressed as $R=R_1R_2$. However, for the parallel concatenation the overall code rate can be calculated as in (2.18), which is greater than R_1R_2.

$$R = \frac{R_1 R_2}{R_1 + R_2 - R_1 R_2} \qquad \textbf{(2.18)}$$

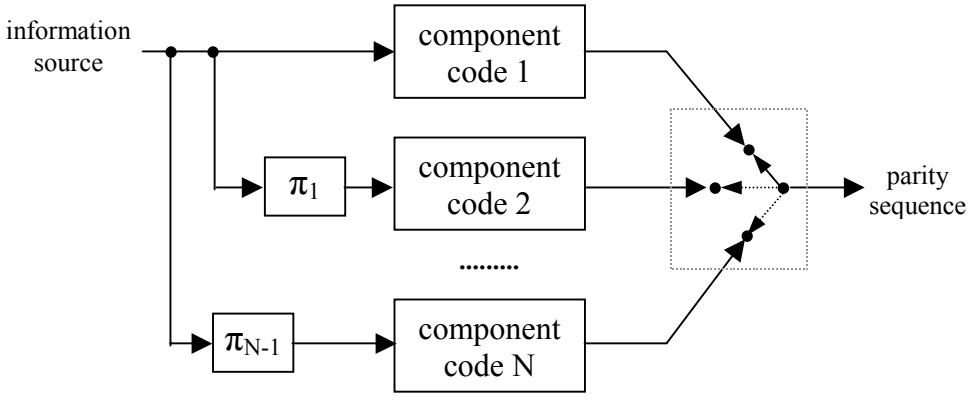


Figure 2.9 Parallel concatenation

2.2.3.3. Soft and Hard Demodulators

In communication systems, the demodulator at the receiver side can either provide binary (hard output) or quantized (soft output) voltage levels to the decoder block that follows it. In this section, the terms *hard* and *soft* output will be explained using the binary phase shift keying (BPSK), which is the typical modulation scheme employed in turbo coding [Massey, 92] .

The BPSK scheme uses two sinusoidal waveforms, s_1(t) and s_2(t), to represent the two binary symbols 1 and 0, respectively. The two signals, s_1 and s_2 are defined as in (2.19), where f_c is the frequency of the carrier, and A denotes the maximum amplitude.

$$s_1(t) = A\cos(2\pi f_c t),\ s_2(t) = -A\cos(2\pi f_c t) \qquad \textbf{(2.19)}$$

Note that s_1 and s_2 have 180^0 phase difference between them, and are often referred to as *antipodal* signals. A more common representation of the sinusoidal functions in (2.19) is obtained by expressing the amplitude in terms of energy [Sklar, 88] , which will be briefly explained next.

The maximum amplitude, A can be expressed as in (2.20) in terms of the root-mean-square amplitude, A_{rms}.

$$A = \sqrt{2}A_{rms} = \sqrt{2A_{rms}^2} \qquad \textbf{(2.20)}$$

The amplitude expression in (2.20) can be rewritten in terms of the average power P (normalised to 1Ω), as indicated in (2.21).

$$A = \sqrt{2P} \qquad \textbf{(2.21)}$$

Note that the average power is equal to the energy delivered per unit time. If the signal duration is T seconds, and E denotes the average energy delivered in T seconds, the maximum amplitude can be stated in terms of E, as shown in (2.22).

$$A = \sqrt{\frac{2E}{T}} \qquad \textbf{(2.22)}$$

The BPSK demodulator can be implemented as a matched filter or as a correlation receiver. It should be noted that these demodulation techniques are considered to be equivalent, and therefore in this section only the correlation receiver is going to be explained. Further information on matched filters can be found in [Sklar, 88] , [Otung, 01] , [Haykin, 94] , [Cooper & McGillem, 88] , and [Carlson, 86] .

The BPSK demodulator can be implemented by using two correlation receivers, as shown in Figure 2.10. First, the received channel signal, *r(t)*, is multiplied by the two reference signals, $s_1(t)$ and $s_2(t)$. Then, the two products are integrated over the symbol period *T*. After sampling the integrator output at time *T*, the two output values $x_1(T)$ and $x_2(T)$ are obtained. The decision on whether $r(t)=s_1(t)$ or $r(t)=s_2(t)$, is made according to (2.23). Note that for the noise-free reception, the $x_1(T)$ and the $x_2(T)$, are equal to *E* and *–E*, respectively. Conventionally, *E* is normalized to 1, and therefore '1' and '-1' represent the transmission of bits '1' and '0', respectively.

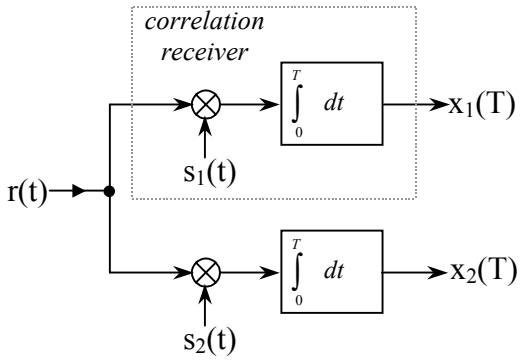


Figure 2.10 Correlation demodulator for BPSK

$$r(t)=\begin{cases} s_1(t)\ , \text{ if } |x_1(T)|> |x_2(T)| \\ s_2(t)\ , \text{ if } |x_2(T)|> |x_1(T)| \end{cases} \qquad \textbf{(2.23)}$$

When the BPSK modulated signals are transmitted over the AWGN channel, the output of the demodulator will be normally distributed around two average values, i.e. –1 and 1. Figure 2.11 shows the two pdf functions, $f_1(x)$, and $f_2(x)$ for the demodulated signals $x_1(T)$ and $x_2(T)$, respectively. As the figure clearly illustrates, the demodulator output is Gaussian in nature with non-zero average. Notice that the horizontal axis is continuous and has infinite number of elements. In practice, due to the memory and numerical representation limitations, the demodulator output is *quantized* as illustrated in Figure 2.12.

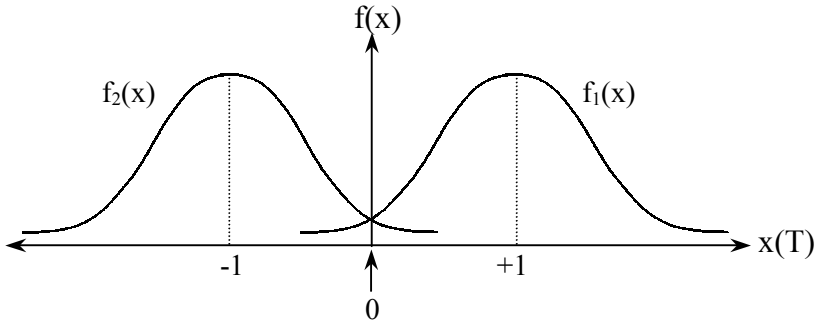


Figure 2.11 Probability distribution of the BPSK demodulator output with AWGN

It should be noted that in Figure 2.12 only 8 evenly spaced discrete values represent the entire x-axis. All x(*T*) values between 0 and 0.2 take 0.2 as their quantized value. Similarly, x(*T*) values less than or equal to –1.4 take –1.4 as their new demodulated value. Introducing more quanta levels can increase the resolution of the horizontal axis.

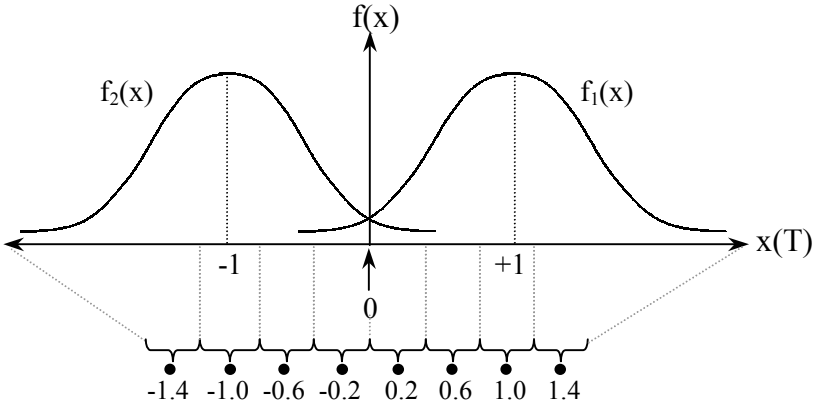


Figure 2.12 Quantization of the noisy BPSK demodulator output

If there were only two quanta levels, then the positive and negative values would be mapped to +1, and –1, respectively. As it was explained earlier, +1 and –1 is mapped to 1, and 0 bit values (or *hard* values). In other words, the only information needed for determining the bit values from the demodulated signal is the polarity or the *sign* of the demodulator output. The magnitude of the output does not change the bit values at all. Therefore, such demodulators are often called *hard output* demodulators.

However, when the quanta levels are greater than two, the magnitude of each level also provides *likelihood* information besides a binary output. Such demodulators are called *soft output* demodulators. The *softness* of the demodulator output comes from the variable degree of certainty, inherent to each quanta level. In other words, each quanta level provides two different types of information; one is the hard value (or the bit value)

determined by the sign, and the other is the *degree of confidence* on that hard value determined by the magnitude.

As it will become clearer in section 2.2.5, the information provided by the soft output demodulators constitute the backbone of the revolutionary iterative decoding. Further information on soft and hard output is also available in [Heegard & Wicker, 99] .

2.2.4. Iterative Encoding

As introduced in 1993 by Berrou [Berrou et al, 93] the turbo encoder constitutes two identical turbo encoders concatenated in parallel with an interleaver, denoted π, that separates them (Figure 2.13). Each component encoder generates a parity sequence, denoted p, which is the same length as the input information sequence u. Therefore, without puncturing the code rate is 1/3, which can be increased by selectively discarding part of the parity sequences p_1 and p_2. If u appears as a part of the codeword, the turbo code is said to be systematic.

It is also possible to increase the number of component encoders, provided that each encoder processes a different interleaved version of the information sequence u. The cost of using such an encoding technique is the reduction in the unpunctured code rate, and the increased decoder complexity, as it will be explained in section 2.2.5.

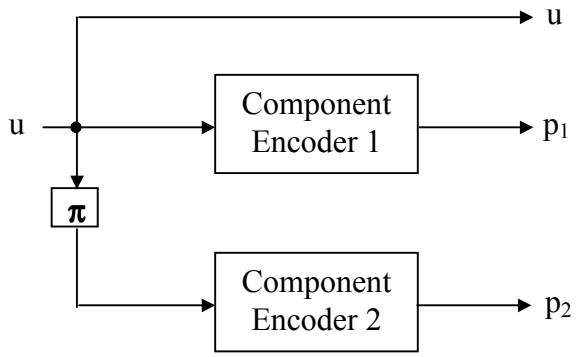


Figure 2.13 Parallel concatenated turbo encoder with two component codes

When identical component codes are used in a turbo encoder the structure is termed as *symmetric*. However, it is also possible to concatenate different codes in the turbo encoder. Such turbo codes are called *asymmetric* turbo codes and for various constituent codes, they have been shown to achieve lower bit error rates than the symmetric designs [Takeshita et al, 99] , [Massey & Costello, 00] .

In this section, iterative coding using convolutional constituent codes will be discussed. However, block codes can also be used as constituent codes in turbo coding [Hagenauer et al, 96] , [Pyndiah, 98] , [Sklar, 97] .

Let us first consider Figure 2.14, where a symmetric turbo encoder has been presented. Each one of the two constituent convolutional codes is the g(7,5) RSC with 4 states (K=3). The component codes are called *recursive systematic convolutional* (RSC) codes, which are important building blocks of turbo encoders. The parity sequences generated by each encoder can be punctured in order to increase the code rate.

One significant point to note in this turbo encoder is the feedback connection that makes each constituent code *recursive*. The recursive convolutional codes have an infinite impulse response (IIR), which plays an important part in the excellent error performance of turbo codes. As mentioned by Benedetto and Montorsi [Benedetto & Montorsi, 95] , the choice of recursive convolutional codes introduces an interleaver gain in the case of parallel concatenation, which can be shown easily.

If w_{min} denotes the minimum weight of an information error event, for an interleaver length of N, there are $C(N,w_{min})$ (2.24) number of ways that such an error event can exist in the same information frame.

$$C(N, w_{\min}) = \frac{N!}{w_{\min}!(N - w_{\min})!} \qquad \textbf{(2.24)}$$

Then under uniform interleaving, the probability of error events with weight w_{min}, can be expressed as in (2.25), where the rightmost expression is valid if $N>>w_{min}$.

$$\Pr = \sum_{i=1}^{N} \frac{1}{C(N, w_{\min})} \approx (w_{\min}!) N^{1-w_{\min}} \qquad \textbf{(2.25)}$$

It can clearly be seen that the interleaver gain in parallel-concatenated convolutional codes is proportional to $N^{1-w_{min}}$. For recursive encoders as w_{min} is always greater than one [Benedetto & Montorsi, 95] , the error event probability decreases with the increasing interleaver size.

Also, Divsalar and Pollara [Divsalar & Pollara, 95] have pointed out that when the codes are non-recursive, the low-weight codeword at the output of one encoder generated by a low-weight information sequence always appears at the output of the other encoder, for any choice of interleaver. This is another motivation for using recursive codes in turbo encoders.

Besides the recursive feature of turbo codes, in the literature using systematic codes has been recognised as a key factor in their excellent error performance.

It is known that the low-weight codewords make the largest contribution to the codeword error of turbo codes, especially at high signal-to-noise ratios. In his analysis, Valenti [Valenti, 96] has shown that using systematic turbo codes increases the minimum codeword weight, as the information sequence becomes a part of the output codeword, which reduces the error probability. Using systematic codes also simplifies the weight distribution analysis, as the weight of the information sequence is helpful in the calculation of the upper limit of codeword errors.

Other authors have emphasized the attractiveness of using systematic turbo codes for practical reasons. As stated by Berrou and Glavieux [Berrou & Glavieux, 96] , for code rates greater than 2/3, finding recursive systematic codes that outperform the non-systematic codes is easier.

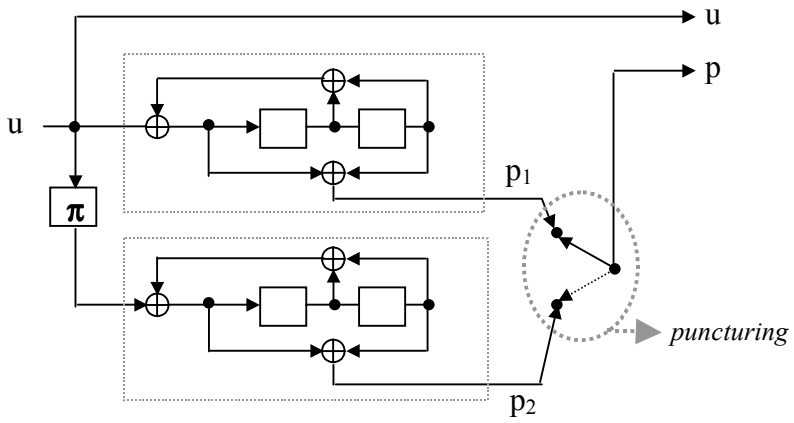


Figure 2.14 Parallel concatenated RSC encoder with g(7,5) component code

The interleaver that separates the two constituent codes in the turbo encoder in Figure 2.14, is crucial in iterative coding. In section 2.2.3.1, interleaver types and the description of interleaving were briefly discussed. In this section, the function of interleaving, which goes beyond a simple re-ordering of information bits, will be explained.

One role of the interleaver in turbo coding is to reduce the correlation between the original information sequence and the interleaved information sequence so that the parity sequences generated by the component encoders are statistically independent. As it will become clear in section 2.2.5, when the parity sequences are uncorrelated, the iterative decoding performs well and fast convergence to the maximum likelihood solution is achieved.

Another function of the interleaver is shaping the weight distribution of the turbo codes. Costello, Hagenauer, Imai and Wicker [Costello et al, 98] have stated that the interleaver has the effect of matching low-weight parity sequences, generated by the low-weight information sequences, with the higher weight parity sequences in almost all cases. More importantly, even for turbo codes with small free distance, minimizing the number of low-weight codewords helps to achieve error performance close to the Shannon limit for large block lengths.

The weight-shaping role of the interleaver has also been previously pointed out by Battail [Battail, 98] . In his work, Battail has also mentioned that interleaving, when combined with concatenation, results in the convolution product of the weight distribution of the component codes.

The trellis representation of each component encoder presented in Figure 2.14, is shown in Figure 2.15. Each branch on the 4-state trellis is marked with two bits, the first of which corresponds to the information bit, and the second is the generated parity bit. After encoding N information bits, the trellis is forced to the zero state, which is often referred to as the *trellis termination*. Trellis termination introduces a certain number of redundant bits, which are called the *tail bits*. The number of generated tail bits per component encoder (N_{tb}) depends on the constraint length, K, of the convolutional code used, and is given by (2.26), where N_{bb} denotes the number of bits per branch. For the turbo encoder in Figure 2.14, as K=3 and N_{bb}=2, each component code generates 4 tail bits.

$$N_{tb} = (K-1)N_{bb} \qquad \textbf{(2.26)}$$

Due to the trellis termination, the RSC turbo codes are not purely convolutional codes, as with convolutional codes each codeword has infinite length. In the literature, due to the finite codeword length, these turbo codes are regarded as having a linear block code structure [Ryan, 97] , [Valenti, 96] .

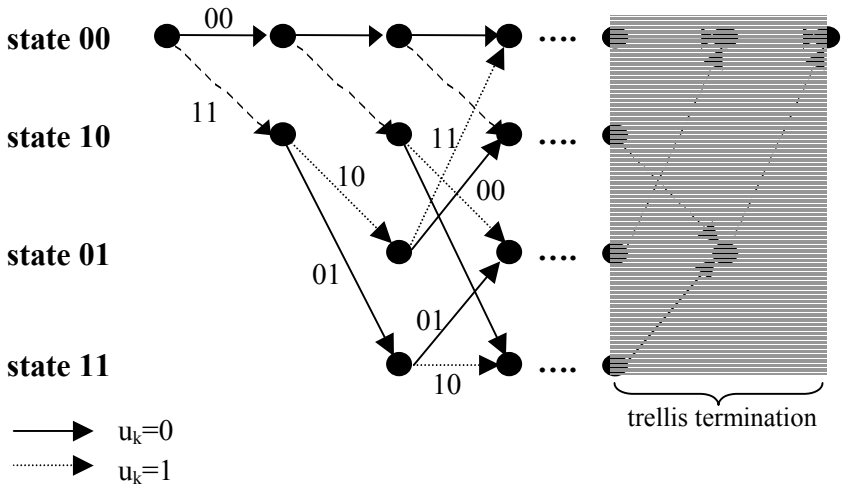


Figure 2.15 Trellis of the RSC g(7,5) turbo code

2.2.5. Iterative Decoding

In this section, the principles of iterative decoding are going to be discussed. First, the concept of 'likelihood' will be introduced via Bayes' theorem. Then, the log-likelihood ratios, which are often used in communications, will be discussed, and their association with soft input/soft output decoding will be emphasised. The section will conclude with the analysis of the famous MAP algorithm, where log-MAP and max-log-MAP concepts will also be introduced.

2.2.5.1. Retrospect of Likelihoods

In order to understand the underlying framework of likelihoods, one needs to go back to the Bayes' theorem, which is used for hypothesis testing.

For two events A and B, the joint probability P(A,B) can be expressed in terms of the conditional probabilities, P(A|B) and P(B|A), as in (2.27).

$$P(A,B) = P(A \mid B)P(B) = P(B \mid A)P(A) \qquad \textbf{(2.27)}$$

Rearranging (2.27) gives (2.28), as shown below.

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)} \qquad \textbf{(2.28)}$$

In communications applications, in the presence of AWGN, (2.28) is expressed in terms of the continuous random variable x, as shown in (2.29). Note that, p and P represent the Gaussian probability density function and discrete event probability, respectively.

$$P(u=i \mid x)=\frac{p(x \mid u=i)P(u=i)}{p(x)} \quad \text{and } i=1,2,....,M \qquad \textbf{(2.29)}$$

The variable u denotes the transmitted data signal, i represents one of M signal candidates, and x is equal to u with added noise. The denominator of (2.29) is equal to the sum of all the possible numerator values over the entire signal set, and is calculated as in (2.30).

$$p(x)=\sum_{i=1}^{M} p(x \mid u=i)P(u=i) \qquad \textbf{(2.30)}$$

Let us now consider data transmission using the BPSK, where voltages –1 and +1, represent the demodulated logic 0 and 1 values, respectively. In order to predict the transmitted u value by looking at the received noisy x, the *maximum a posteriori* (MAP) decision rule based on the conditional probabilities can be used as follows.

if *P(u=+1|x) > P(u=-1|x)* then decide *u*=+1

if *P(u=+1|x) < P(u=-1|x)* then decide *u*=-1

The above decision rules can also be expressed as in (2.31).

$$\begin{aligned} &\text{If } \frac{P(u=+1 \mid x)}{P(u=-1 \mid x)}>1 \text{ then } u=+1 \\ &\text{If } \frac{P(u=+1 \mid x)}{P(u=-1 \mid x)}<1 \text{ then } u=-1 \end{aligned} \qquad \textbf{(2.31)}$$

When the conditional probabilities in (2.31) are replaced with their equivalents in (2.29), the *likelihood ratio test* can be obtained as in (2.32).

$$\begin{aligned} &\textbf{If } \frac{p(x \mid u=+1)P(u=+1)}{p(x \mid u=-1)P(u=-1)}>1 \text{ then } u=+1 \\ &\textbf{Else } u=-1 \end{aligned} \qquad \textbf{(2.32)}$$

For iterative decoding, the likelihood ratio test is transformed into a more useful form, namely the log-likelihood ratio, which is suitable for representing the soft decision at the output of a soft decision demodulator [Sklar, 97] .

2.2.5.2. Log-Likelihood Ratio and the Iterative Decoders

The log-likelihood ratio, $L(u|x)$, in (2.33) is obtained by taking the natural logarithm of the left side of the inequality given in (2.32). Note that if $L(u|x)$ is positive then u=+1, and otherwise u=-1, as the logarithm of the right side is negative.

$$L(u \mid x)=\log\left[\frac{p(x \mid u=+1)P(u=+1)}{p(x \mid u=-1)P(u=-1)}\right] \qquad \textbf{(2.33)}$$

Using logarithmic identities, L(u|x) can be stated as in (2.34).

$$L(u \mid x) = \log\left[\frac{p(x \mid u=+1)}{p(x \mid u=-1)}\right] + \log\left[\frac{P(u=+1)}{P(u=-1)}\right] \qquad \textbf{(2.34)}$$

The expression in (2.34) can be simplified to (2.35). The log likelihood ratio (LLR) for u at the demodulator output is denoted $L`(u)$. The second term, $L_c(x)$, depends on the channel measurements made at the demodulator. The last term is called the *a priori* LLR of the symbol u, and depends on the probability of transmitting either +1 or –1.

$$L`(\mathrm{u}) = L_c(\mathrm{x}) + L_u(\mathrm{u}) \qquad \textbf{(2.35)}$$

Besides the LLRs provided at the demodulator output, the decoder also introduces a soft information term called the *extrinsic* LLR to (2.35). The extrinsic LLR, denoted $L_e(u)$, is calculated from the transmitted parity information. For a systematic code, the LLR at the output of the decoder, denoted $L(u)$, can be expressed as in (2.36)

$$L(u) = L`(u) + L_e(u) = L_c(x) + L_u(u) + L_e(u) \qquad \textbf{(2.36)}$$

The $L(u)$ is also called the *a posteriori* LLR, and is used for hard decision as well as a measure of reliability. If the sign of $L(u)$ is negative, then the transmitted u is –1, and for positive values of $L(u)$, u is decided as +1. In addition, the magnitude of $L(u)$ indicates how reliable the binary decision is; the higher the magnitude, the more reliable the decision becomes.

Until now, the essential LLRs for performing iterative decoding have been discussed. In order to get a clearer picture of how the iterative decoding works, the soft input-soft output (SISO) decoder structure will be explained using the LLRs, in the rest of this section. Let us first consider the SISO component decoder (Figure 2.16), which is the building block of a turbo decoder.

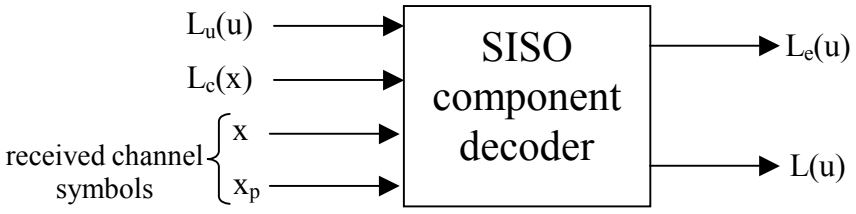


Figure 2.16 SISO component decoder

The component decoder accepts the $L_u(u)$ and the $L_c(x)$ LLRs as the input and delivers $L_e(u)$ and the $L(u)$ at the output. Note that the received noisy information, x, and the parity, x_p, symbols are also fed to the decoder. To perform iterative decoding, at least two SISO component decoders are needed. The number of component encoders used at the

transmitter side determines the number of component decoders in a turbo decoder. By connecting a number of such SISO decoders in parallel or series, a turbo decoder can be realised.

As an example, let us consider the serial turbo decoder shown in Figure 2.17, which corresponds to the encoder structure presented in Figure 2.14. Note that π denotes interleaving, and when it is used as a subscript it indicates the interleaved version of a symbol sequence.

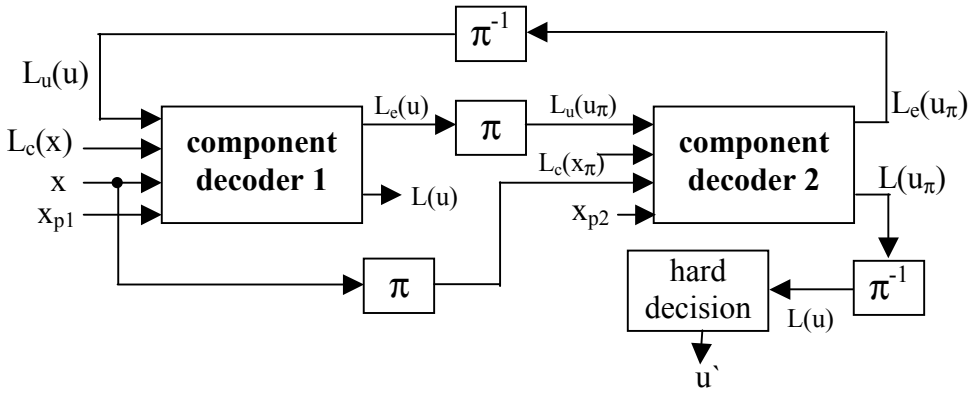


Figure 2.17 Serial turbo decoder

One important thing to note about the two component decoders in Figure 2.17, is that they are separated by an interleaving stage. Similar to the encoder, the first component decoder processes the original received information symbols, whereas the second decoder accepts the interleaved version of the same sequence. Note that the parity symbols fed to the two decoders are also different. When both component decoders process the a priori information for each information symbol, one *iteration* is completed. In turbo decoding the extrinsic information of one decoder becomes the a priori information for the other. Each iteration allows a component decoder to refine its previous extrinsic information by incorporating the soft output of the other component to the new LLR calculation. During iterations the a posteriori output of the first component decoder is not fed to the second one, as shown in Figure 2.17. Iterations continue until the decoding operation does not significantly improve the LLRs anymore. After the completion of iterations, the a posteriori output of the second component decoder is de-interleaved and fed to the hard decision block, which assigns a binary value, *u`*, to each information symbol.

Turbo decoding can also be performed in a parallel fashion as illustrated in Figure 2.18. When the parallel and the serial structures are compared, a number of operational differences are noticed.

In parallel decoding, there are two LLR pipelines that are active simultaneously. Therefore, after a single iteration with a parallel decoder, both component decoders are used twice, whereas in serial turbo decoders, each component decoder is activated only once. The hard decision stage of a parallel decoder compares the a posteriori information of both component decoders, and the maximum LLR is selected. However, hard decision is applied to the a posteriori output of the second component decoder, in serial decoding.

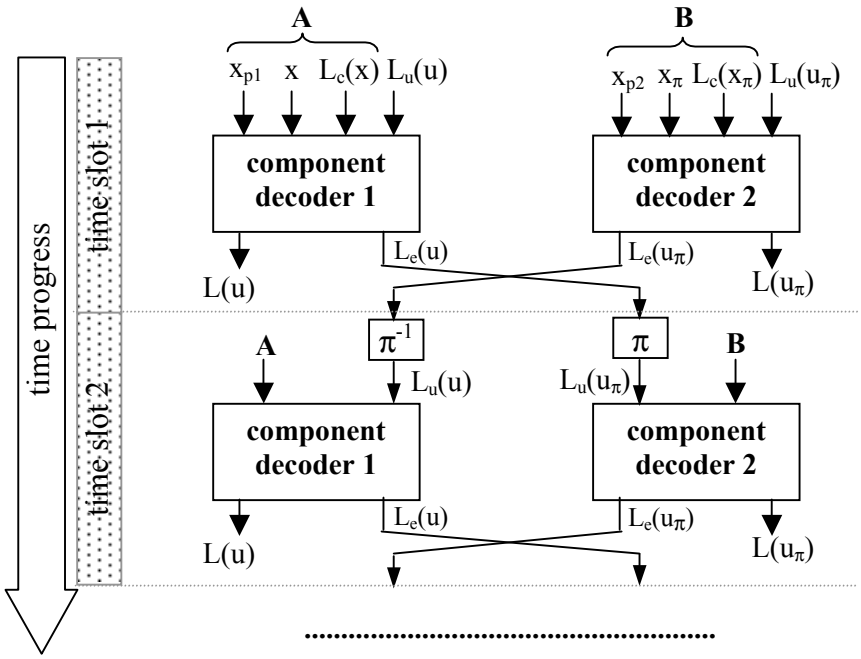


Figure 2.18 Parallel turbo decoder

Using parallel decoding is more advantageous compared to the serial one [Heegard & Wicker, 99] because of the following reasons:

- ***Efficient use of hardware-*** In parallel decoding, both component decoders operate simultaneously, which means that the decoder hardware resources are efficiently allocated, whereas in serial decoding, one component decoder has to wait until the output of the other becomes available.
- ***Unbiased operation-*** In parallel decoding, both component decoders are initialised with identical a priori input, which is zero. Therefore, the generated LLRs after half iteration are almost completely independent. In serial decoding the second component decoder uses the extrinsic output of the first one as its initial a priori input, and consequently the LLRs generated by the second component decoder are inherently biased towards the first one. In other words, as the soft information contributions of the component decoders do not become dominant during iterations, parallel decoding is a means of providing *equal opportunity* for both component decoders in iterative decoding.

Up to this point, the principles of iterative decoding have been covered while giving insights into serial and parallel turbo decoding techniques. In the next section, the SISO decoding algorithm, which is used in the calculation of the LLRs within the component decoders, will be discussed in detail.

2.2.5.3. The Maximum A Posteriori Probability Algorithm

Although other SISO decoding algorithms such as the soft output Viterbi algorithm (SOVA) [Hagenauer & Hoeher, 89] and [Hagenauer et al, 96] are often used in iterative decoding, it would not be wrong to say that the maximum a posteriori (MAP) probability algorithm, originally introduced as the *BCJR algorithm* by Bahl et al [Bahl et al, 74] , has earned the credit the turbo codes have today. Despite its high numerical complexity and large storage requirements, the MAP algorithm is still an indispensable part of turbo decoding due to its excellent performance. So far, a significant amount of research time has been dedicated to developing simplified versions of this powerful decoding technique [Gross & Gulak, 98] , [Liu et al, 00] , as well as implementing and analysing the performance of the existing sub-optimal MAP algorithms [Robertson et al, 95] , [Pietrobon, 98] , [Viterbi, 98] .

In this section, the details of the MAP algorithm will be discussed, and the sub-optimal variations of this algorithm are going to be mentioned where relevant.

For the terminated convolutional codes, the MAP algorithm calculates the probabilities in (2.33) by using the trellis information, which is *all possible state transitions* for *all possible transmitted information bits*, as well as *all possible* codeword*s* for *all possible state transitions*. In order to make use of the trellis information in the calculation of the LLR in (2.33), first a set of probability metrics need to be defined.

Let us consider how the trellis information at any depth can be interpreted in terms of joint probabilities. The k^{th} depth of a binary trellis is illustrated in Figure 2.19, where the initial and the final state for that depth are denoted as $s`$ and s, with respect. Assuming that there are two symbols per branch, the likelihood that a received symbol couple $\{x_k,x_{kp}\}$, will be decoded correctly as the transmitted couple $\{u_k,p_k\}$, is associated with the state transitions at depth k. This dependency can conveniently be expressed as a joint probability, $p(s`,s,X)$, where X denotes the received symbol vector.

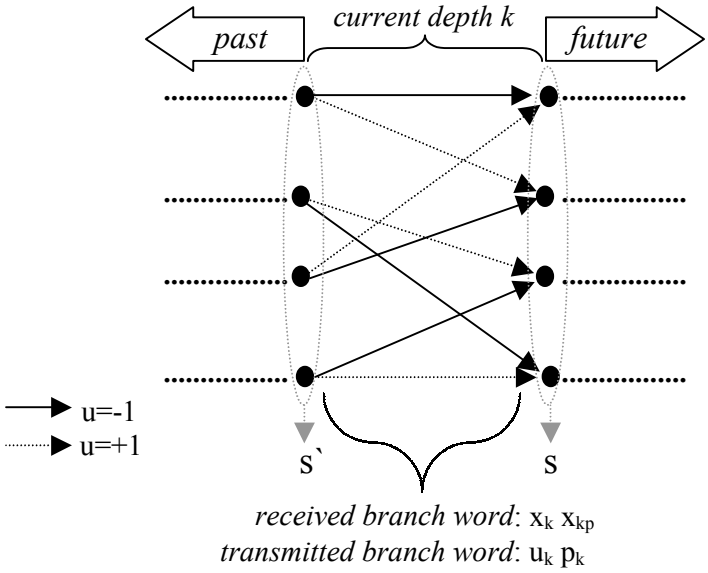


Figure 2.19 One trellis depth

As for each trellis depth, the transmitted information symbol, u_k, can take one of two values, the LLR will be the ratio of the sum of joint probabilities for u_k=+1 and u_k=-1 calculated for all state transitions. Therefore, using the joint probability, *p(s`,s,X)*, (2.33) can be expressed as in (2.37), which is the a posteriori output of a component decoder.

$$L(u_k \mid X) = \log\left[\frac{\sum_{u_k=+1,(s`,s)} p(s`,s,X)}{\sum_{u_k=-1,(s`,s)} p(s`,s,X)}\right] \qquad \textbf{(2.37)}$$

The next step of the analysis is the calculation of *p(s`,s,X)*, which can be expressed as the product of three probabilities, namely α, β, and γ (2.38), for a memoryless channel. Each product term can be described as follows:

***$\alpha_k(s`)$*** *is the probability of reaching state s` at depth k, starting from the beginning of the trellis, and moving towards the end of the trellis.*

$\beta_k(s)$ *is the probability of reaching state s at depth k, starting from the end of the trellis, and moving towards the beginning.*

***$\gamma_k(s`,s)$*** *is the branch probability of transition from s` to s at depth k.*

$$p(s`,s,X) = \alpha_k(s`)\gamma_k(s`,s)\beta_k(s) \qquad \textbf{(2.38)}$$

Due to the way the α and β probabilities are calculated, their computations are sometimes referred to as the *forward* and *backward recursions*, respectively. The recursive calculation can be easily understood when visualized on a trellis. As an example, consider Figure 2.20, where a 4-state terminated trellis with *N*+2 depths is shown. Each state on the trellis has an associated α and a β probability, and each branch is assigned a γ probability.

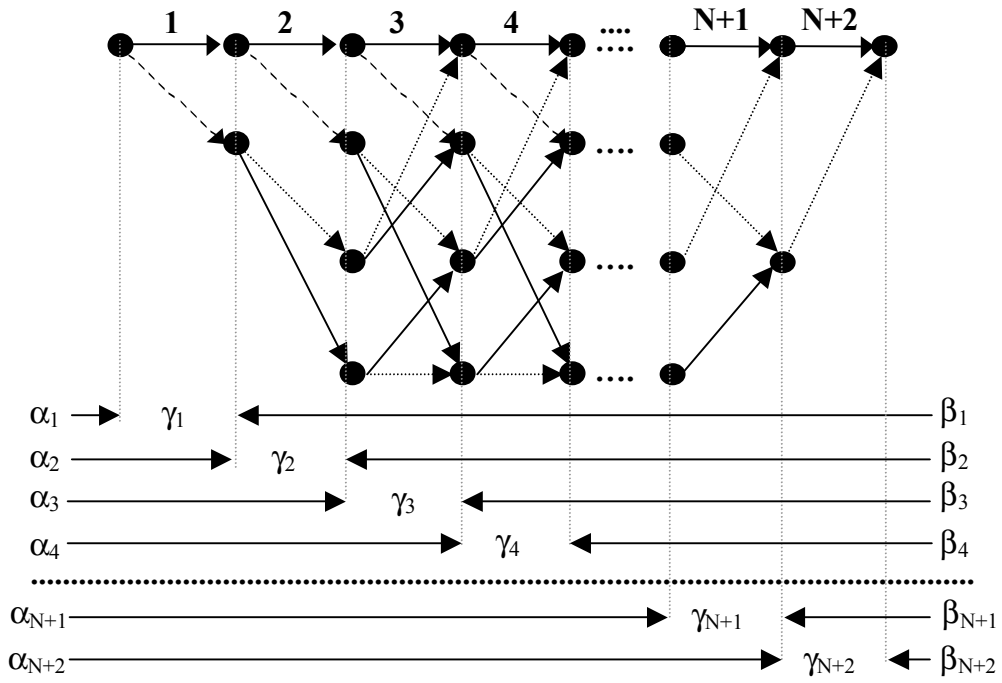


Figure 2.20 Calculation of α, β, and γ probabilities

The α and β probabilities are calculated in a recursive fashion, as expressed in (2.39) and (2.40).

$$\alpha_k(s) = \sum_{s`} \gamma_k(s`,s)\alpha_k(s`) \qquad \textbf{(2.39)}$$

$$\beta_k(s`) = \sum_{s} \gamma_k(s`,s)\beta_k(s) \qquad \textbf{(2.40)}$$

Each state transition from $s`$ to $s$, depends on the probability of transmitting information symbol, $u_k$, denoted $P(u_k)$, and the conditional probability of receiving a codeword vector, $X$, given that $u_k$ is transmitted, denoted $p(X|u_k)$. Therefore, $\gamma_k(s`,s)$, can be expressed as in (2.41).

$$\gamma_k(s`,s) = p(X \mid u_k)P(u_k) \qquad \textbf{(2.41)}$$

The conditional probability $p(X|u_k)$ in (2.41) can be expressed as in (2.42) [Ryan, 97] , [Hagenauer et al, 96] . Notice that the number of parity bits per information bit is assumed to be n. Therefore, the sum term applies to n received parity symbols and the parity symbols on a trellis branch. The B_k term is a constant, which cancels out when $\gamma_k(s`,s)$ is used in a LLR. The channel reliability factor, denoted L_c, is equal to $4E_s/N_o$, where E_s is the energy per symbol [Ryan, 97] , [Hagenauer et al, 96] . When E_s is normalized to 1, L_c becomes $2/\sigma^2$, where σ^2 denotes the noise variance [Sklar, 97] . This can easily be shown by expanding the $L_c(x)$ term in (2.34), which will not be explained here.

$$p(X \mid u_k) = B_k e^{\left(\frac{1}{2} L_c x_k u_k + \frac{1}{2} L_c \sum_{v=1}^{n} x_{kp,v} p_{k,v} \right)} \qquad \textbf{(2.42)}$$

The second product term in (2.41) is similar to the probabilities in the a priori LLR mentioned previously in (2.34). In fact, $P(u_k=\pm 1)$ can be expressed as in (2.43), using the $L_u(u)$ term in (2.34). Again note that the constant term, A_k, cancels out when expressed in the LLR.

$$P(u_k = \pm 1) = \left(\frac{e^{-L_u(u_k)/2}}{1 + e^{-L_u(u_k)}} \right) e^{(L_u(u_k) u_k / 2)} = A_k e^{(L_u(u_k) u_k / 2)} \qquad \textbf{(2.43)}$$

By multiplying the expressions in (2.42) and (2.43), $\gamma_k(s`,s)$ can be obtained as in (2.44). As the second exponential term depends only on the parity information, it is often called the *extrinsic gamma*, and is denoted $\gamma_{ek}(s`,s)$.

$$\gamma_k(s`,s) = e^{\left(\frac{1}{2} u_k [L_c x_k + L_u(u_k)]\right)} e^{\left(\frac{1}{2} L_c \sum_{v=1}^{n} x_{kp,v} p_{k,v}\right)} \qquad \textbf{(2.44)}$$

When all the probability terms obtained so far are substituted into (2.37), the a posteriori expression for the MAP algorithm can be expressed as in (2.45). As it can be seen in (2.36), the a posteriori probability expression derived for the MAP algorithm is in the same form used in iterative decoding.

$$L(u_k) = L_c(u_k) + L_u(u_k) + \log \left[\frac{\sum_{u_k=+1,(s`,s)} \gamma_{ek}(s`,s) \alpha_k(s`) \beta_k(s)}{\sum_{u_k=-1,(s`,s)} \gamma_{ek}(s`,s) \alpha_k(s`) \beta_k(s)} \right] \qquad \textbf{(2.45)}$$

The optimal MAP algorithm discussed so far, suffers from high numerical complexity, which is primarily due to the calculation of the exponential functions. The challenge in MAP algorithm is the exact calculation of (2.46), where a_i is a real number.

$$\log(e^{a_1} + e^{a_2} + e^{a_3} + ... + e^{a_n}) \qquad \textbf{(2.46)}$$

By using the Jacobian logarithm [Robertson et al, 95] , [Liu et al, 00] , this problem can be solved. For two a_i values, this logarithm can be expressed as in (2.47), where the second logarithmic term is the *correction term*, and is denoted as $f_c(|a_1-a_2|)$.

$$\log(e^{a_1} + e^{a_2}) = \max(a_1, a_2) + \log(1 + e^{-|a_1 - a_2|}) \qquad \textbf{(2.47)}$$

The expression in (2.47) can be approximated to $max(a_1, a_2)$, which is used by the *max-log-MAP algorithm*. As it can clearly be seen, this approximation eliminates the calculation of any exponential function, and it is a sub-optimal technique. Due to its reasonable

computational complexity and high error correction performance, the max-log-MAP algorithm is a popular choice in SISO decoding [Chass & Gubeskys, 00] . When the correction term is included with the maximization function, the algorithm is called the *log-MAP*, which provides higher accuracy than the max-log-MAP in the LLR calculations at the expense of higher computational overhead. It should be noted that the expression in (2.46) could be calculated recursively, by using (2.47), which has been proven by Robertson et al [Robertson et al, 95] .

2.3. Automatic Repeat Request (ARQ)

Automatic Repeat Request is an error-control technique that is based on the detection of frame errors rather than their correction. The receiver informs the transmitter about the outcome of the error detection by sending either a positive or a negative acknowledgement over the return channel. Once the transmitter receives the acknowledgement information, it retransmits the frame according to the ARQ protocol employed.

In this section the common ARQ techniques are discussed under two sub-sections, which are the Basic ARQ Schemes, and the Hybrid ARQ Schemes.

2.3.1. Basic ARQ Schemes

There are three basic ARQ schemes: the *stop-and-wait (SAW)*, *go-back-N (GBN)* and *selective-repeat (SR)* ARQ, [Lin et al, 84] , [Sklar, 88] . Operation of each scheme will be briefly explained in this section.

In SAW, after transmitting a frame, the transmitter waits for acknowledgement (ACK) from the receiver (Figure 2.21). Once an ACK is received, the consecutive frame is transmitted. The receiver sends a negative acknowledgement (NACK), if an error in the frame is detected. As a response to a NACK, the transmitter retransmits the same frame. The SAW scheme is inefficient because there is an idle time between the transmitted frames.

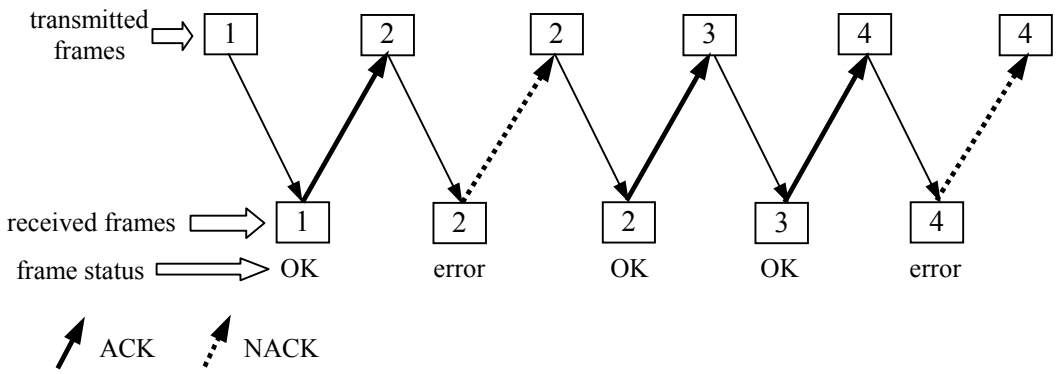


Figure 2.21 Stop-and-wait ARQ protocol

In a GBN scheme (Figure 2.22) the transmitter sends the frames without having to wait for the ACK response, which arrives after a round-trip delay. When a frame is detected in error, the receiver sends a NACK. As the transmitter receives the NACK with a round-trip delay, it needs to resend the most recent *N* frames to the receiver, where *N* denotes the number of frames that are transmitted within the round-trip delay. With the GBN scheme, the inefficiency of SAW technique is overcome by continuously transmitting frames. However, the GBN's inefficiency comes from the retransmission of possibly error-free *N-1* frames, which follow the requested frame. This inefficiency increases as the round-trip delays get longer, and consequently *N* gets larger.

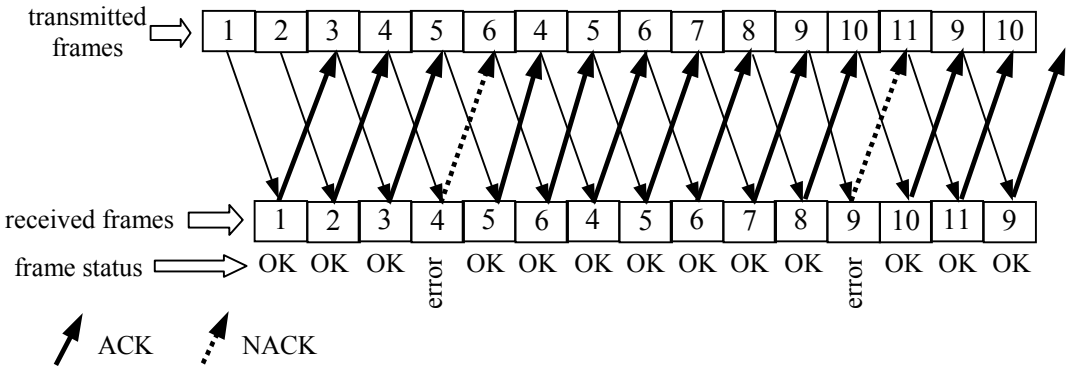


Figure 2.22 Go-back-N ARQ protocol, where *N*=3

The last ARQ scheme to be discussed is the SR (Figure 2.23), which is similar to the GBN. As in the GBN, the frames are continuously transmitted until a NACK is encountered at the receiver. Instead of resending *N* most recent frames, the transmitter only resends the frame, which is detected in error. Because of its a priori knowledge of the round-trip delay and the rate of transmission, the transmitter can determine which frame to retransmit. Transmission of the frames continues from where it stopped before the NACK. In the SR scheme, after sending a NACK, the receiver needs to store the superseding frames in a buffer, so that after receiving the requested frame error-free, the right frame order can be

maintained. Note that the receiver buffer size needs to be sufficiently large to prevent overflow. Even though the SR scheme is the most efficient compared to the SAW and the GBN ARQ systems, its implementation is comparatively more complex [Taub & Schilling, 86] , [Lin & Costello, 83] .

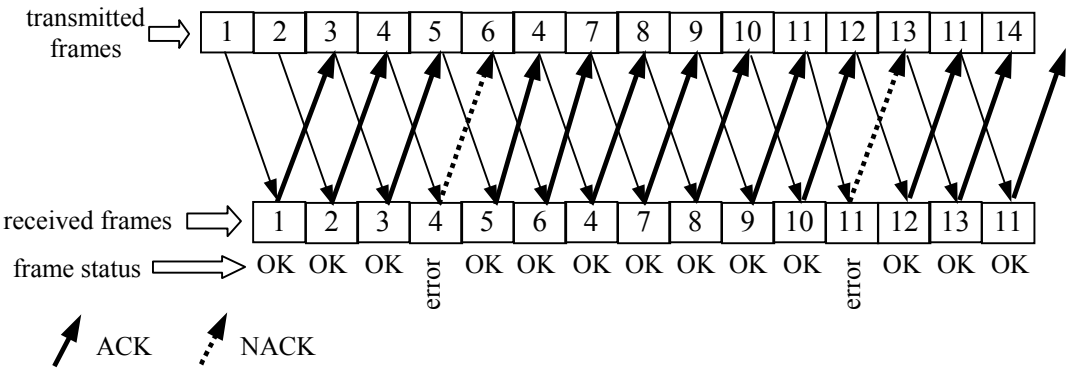


Figure 2.23 Selective-repeat ARQ protocol

2.3.2. Hybrid ARQ Schemes

When the channel noise level is high, using the basic ARQ systems discussed above cause rapid decrease in the information throughput. A transmitted frame might need to be retransmitted almost indefinitely, as it will be received in error almost every time it is retransmitted. In order to increase the system efficiency, a FEC scheme is used with an ARQ system, in which case the overall system is called a *hybrid* [Lin & Costello, 83] , [Lin et al, 84] . There are two types of hybrid ARQ schemes, which are called *Type-I* and *Type –II*, which are going to be explained with examples next.

In the Type-I ARQ, once a decoded frame is detected in error, the receiver discards that frame and sends a NACK to the transmitter. Upon receiving the NACK, the transmitter resends the same frame. Let us consider an example to clarify the Type-I operation. In Figure 2.24, a parity sequence $\{p_1, p_2, p_3, p_4, p_5, p_6\}$ is available at the transmitter side. After puncturing, only the parity bits with odd indices, i.e. $\{p_1, p_3, p_5\}$ are transmitted. At the receiver side, the FEC scheme uses the odd parity bits and decodes the frame. If the decoded frame is still detected as in error, the received parity frame is discarded and a NACK is sent back to the transmitter. As a response, the transmitter resends the exact parity frame, which is again decoded at the receiver, until a positive acknowledgement is received for that frame.

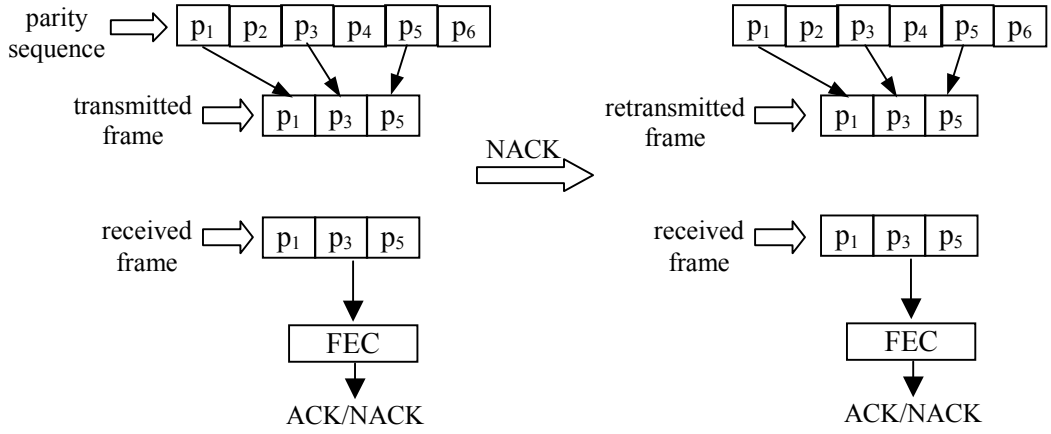


Figure 2.24 An example of Type-I Hybrid ARQ scheme

The Type-II operation varies from the Type-I in that even though a NACK is sent for a frame, the receiver does not discard that frame. After the transmitter sends additional information for the same frame, all the received information is jointly used for decoding that frame. The superiority of the Type-II scheme over the Type-I is that additional side information in the retransmitted frame increases the robustness of the frame against potential channel errors.

The example presented in Figure 2.25 shows how the Type-II scheme works. The first transmission is identical to the Type-I example illustrated in Figure 2.24. After the receipt of a NACK, instead of resending the parity sequence available at the receiver, new parity bits, i.e. parity bits with even indices, are transmitted. During the second decoding, the error correction is with the entire parity sequence, and hence the correction capability is increased.

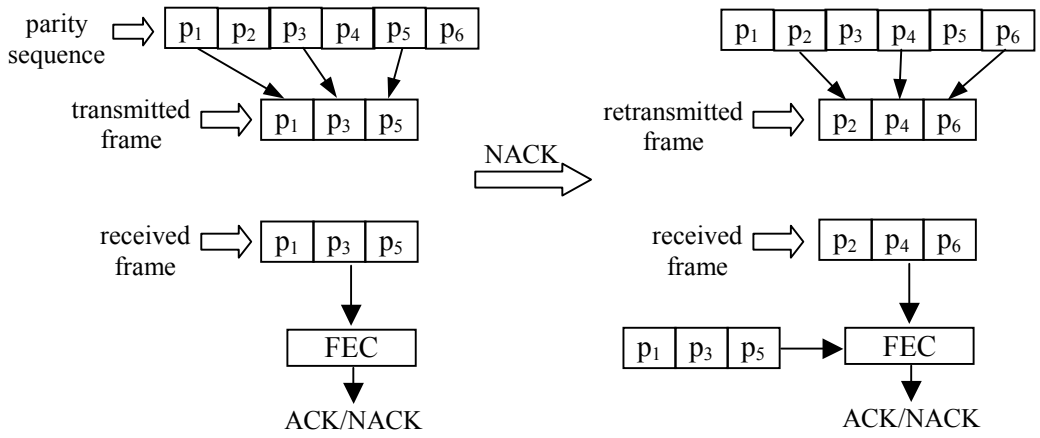


Figure 2.25 An example of Type-II Hybrid ARQ scheme

Chapter 3

Multifold Turbo Codes

3. Multifold Turbo Codes

3.1. Introduction

In the classical turbo coding, originally introduced by Berrou et al [Berrou et al, 93] , each information bit in a frame is encoded twice - one component encoder encodes the information frame, while another encodes the re-ordered, i,e, interleaved, version of the same frame. At the output of each component encoder, one parity bit is generated per encoded information bit.

The traditional turbo decoder constitutes two component decoders, which are separated by an interleaver and a de-interleaver block. These component decoders decode the received information symbols with their corresponding received parity symbols. The statistical independence provided by the interleaver, allows the two component decoders to exchange LLRs to iteratively improve the decoding reliability for each information symbol.

It is known that using stronger component codes and/or better interleaving and decoding strategies can significantly improve the error performance of turbo codes. Recently, a few alternative methods of improving the error performance of turbo codes have been investigated.

Frey and MacKay [Frey & MacKay, 00] have developed *irregular turbo codes*, which encode a set of information bits multiple times in each component encoder. This method requires a decrease in the overall code rate, which can be increased to a ½ or greater by puncturing. They define the *degree* of an information bit as the number of trellis depths, in which it appears. In other words, an information bit of degree *d*, appears in *d* depths and has *d* associated parity bits. In their work, the information bits with high degrees are decoded more reliably, as they receive a large number of likelihood ratios during iterations. By using various degrees obtained from Monte Carlo simulations, the authors have shown that for information frame sizes 131072 and 8920 bits, coding gains up to 0.23 dB and 0.20 dB compared to the classical turbo codes can be achieved, respectively.

In multifold turbo coding [Tanriover et al, 01a] , [Tanriover et al, 01b] the information frame is divided into a variable number of equal sized segments, prior to encoding. Each information bit is encoded multiple times with this novel coding scheme. Because of multiple encoding, each information bit receives multiple likelihood ratios during decoding. One important difference between this coding scheme and the classical one is

that the information frame size is not the same as the interleaver size. As will be discussed in this chapter, significant improvements in error performance can be achieved by using multifold turbo codes with short frame sizes and iterations up to 16.

The organisation of this chapter is as follows. Section 3.2 introduces the multifold information terminology that is used throughout this chapter. The multifold encoding and decoding are explained in section 3.3, where the overall code rate of multifold codes is also covered. One member of the class of multifold turbo codes, namely the two-fold turbo code, is introduced in section 3.4. The encoder and decoder structures for the two-fold codes as well as their error performance and decoding complexity are covered in this section. Section 3.5 concludes the chapter with brief comments on results and multifold turbo codes.

3.2. Multifold Information Terminology

In traditional turbo coding, a block of information bits and its interleaved version are encoded in order to generate two sets of redundancy bits. Each information bit, in this case, is encoded twice and the total number of parity bits is equal to twice the information frame size (N_B). The information frame, in this case, is encoded as a whole.

In multifold coding the information frame is divided into shorter bit groups prior to encoding. Before explaining how this is achieved, the information terminology used in multifold coding is going to be introduced first.

The smallest unit in an information frame is an *information bit* (Figure 3.1). When L_S bits are grouped together, they compose an *information segment* (or segment for short). An information frame can constitute N_S such segments, which have equal lengths. According to this description, the total number of bits in an information frame will be equal to the product of N_S and L_S as in equation (3.1).

By combining N_G segments at a time, a set of information sub-frames are constructed before encoding. Note that N_G needs to be less than N_S for multifold coding and greater than 1 due to iterative decoding requirements. This point is going to be revisited later on.

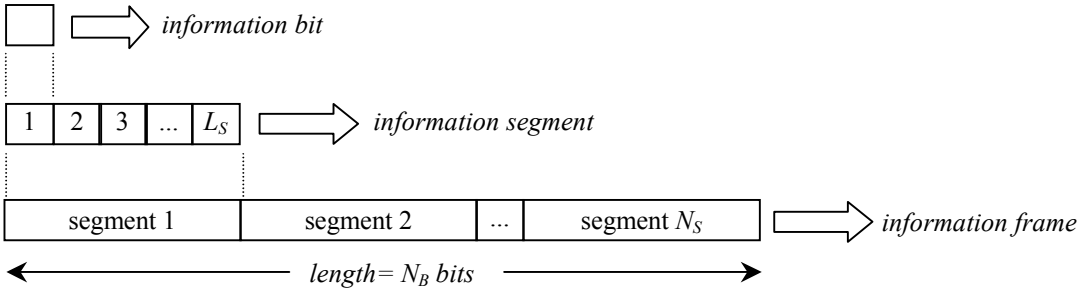


Figure 3.1 Multifold coding information terminology

Once N_S and N_G are determined, the number of all possible information sub-frames (C_S) of an information frame can be calculated from $C(N_S,N_G)$ in (3.2), which is *the number of all possible segment combinations selecting N_G segments from a possible N_S at a time*.

$$N_B=N_S \cdot L_S \qquad \textbf{(3.1)}$$

One example of sub-frame construction is illustrated in Figure 3.2 for N_S=4 and N_G=2. In this example, the total number of sub-frames, i.e. C_S, is 6. Each sub-frame constitutes 2 combined segments and its length is half of the information frame.

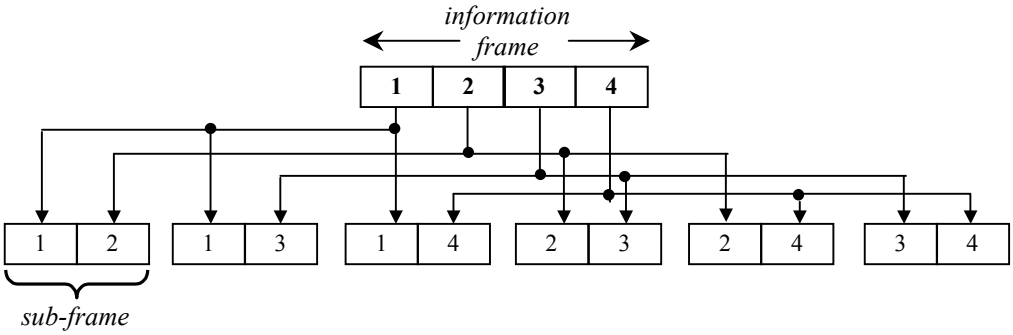


Figure 3.2 Sub-frame construction example for N_S=4 and N_G=2

Considering the definitions above, the information terminology of multifold coding can be summarised with the following three remarks:

- *A segment is a combination of bits, constructed by choosing 1 bit at a time from L_S available bits.*
- *An information frame is a combination of segments, constructed by choosing N_S segments at a time from N_S available segments.*
- *A sub-frame is a special combination of segments, constructed by choosing N_G segments at a time from N_S available segments.*

$$C_S = C(N_S, N_G) = \frac{N_S!}{N_G! \cdot (N_S - N_G)!} \qquad \textbf{(3.2)}$$

In the next section, details of multifold encoding will be discussed.

3.3. Multifold Coding Fundamentals

3.3.1. Multifold Encoder

In multifold coding, each information sub-frame is encoded alone and is separated from other sub-frames via interleaving. First, let us consider an encoding example for the sub-frame construction in Figure 3.2.

Figure 3.3 illustrates the multifold encoding of sub-blocks for N_S=4 and N_G=2. Interleaver blocks are indicated with $\Pi(s_i s_j)$, where s_i and s_j denote different segment indices. Each sub-frame is separated from another by a different interleaving sequence. The turbo encoders are identical in structure and generate sets of parity bits, denoted $P\{s_i s_j\}$, at the output. Since the code is systematic, the information sequence $I\{1234\}$ is also included as part of the codeword.

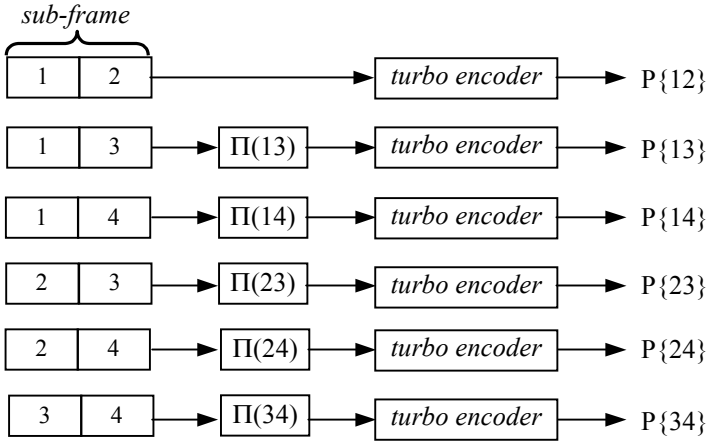


Figure 3.3 Multifold encoding example for N_S=4 and N_G=2

It can be seen from Figure 3.3 that the total number of encoders used is equal to the number of sub-frames. Therefore, in multifold turbo coding, C_S encoders are needed. In addition, the minimum number of interleavers is equal to C_S-1, as one of the sub-frames can be encoded without interleaving.

In multifold coding, each segment is encoded more than once in different sub-frames, which is a key feature of multifold coding. The encoding frequency per segment is represented with M, which defines an *M-fold* turbo code. In the above example, as each segment is encoded 3 times, M is 3, and the scheme is called 3-fold turbo code. The M parameter is a function of N_G and N_S, and can be uniquely determined from those two terms. The final statement will be explained in detail.

Assume that we would like to determine the encoding frequency for one segment, denoted S_i, out of N_S segments. The total number of segments except S_i in this case is equal to N_S-1. Similarly, the number of segments that can be grouped at a time without S_i is N_G-1. Therefore, $C(N_S\text{-}1,N_G\text{-}1)$ will be the number of sub-frames excluding S_i. Since in multifold coding, N_G segments need to be combined, S_i has to be attached to each of $C(N_S\text{-}1,N_G\text{-}1)$ sub-frames. As each sub-frame is fed to an encoder, all bits in S_i are encoded $C(N_S\text{-}1,N_G\text{-}1)$ times. Thus, M is equal to $C(N_S\text{-}1,N_G\text{-}1)$.

The generalised block diagram of an M-fold turbo encoder outlines the information presented in this section (Figure 3.4). Prior to encoding, an information frame with N_S segments, $I\{1,2,3,...,N_S\}$, is re-arranged into sub-frames $I\{C_1\},I\{C_2\},...,I\{C_a\}$, where $1 \le a \le C_S$. Here, $I\{C_i\}$ represents the set of information bits that belong to the segment combination C_i. Using C_S-1 interleavers, the sub-frames are permuted and are encoded to generate parity sequences $P\{C_1\}$, $P\{C_2\},\ldots,$ $P\{C_a\}$. The generated codeword for each information frame also includes $I\{1,2,3,...,N_S\}$, since the code is systematic.

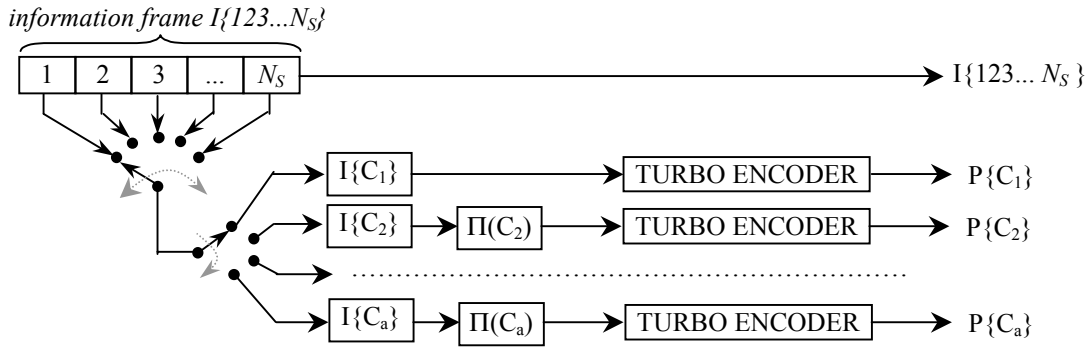


Figure 3.4 Multifold turbo encoder

3.3.2. Code Rate of M-fold Codes

The N_S and N_G parameters of an M-fold turbo code are useful for designing the encoder and the decoder. So far, it has been shown how they can uniquely determine the required number of encoders, and interleavers as well as the multifold index, M.

In this section, the relationship between the N_S and N_G parameters and the mother code rate for systematic M-fold turbo codes will be uncovered.

First, consider Table 3.1, where M values for a set of (N_S,N_G) pairs are given. Note that the special case where N_G=N_S (along the diagonal) corresponds to classical turbo coding, which can also be called *1-fold* coding.

In addition to the above case, when N_G is fixed at 1, for all N_S values, another 1-fold code can be designed. However, with this code iterative decoding cannot be performed since each segment is also a sub-frame in this case. Consequently, the sub-frames share no common information bits, and therefore it is not possible to pass extrinsic information between them during decoding. This point will become clear while describing multifold decoding. Thus, N_G=1 case is not suitable for turbo coding, and is ruled out in Table 3.1.

		N_G									
		2	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***
N_S	***2***	1	×	×	×	×	×	×	×	×	×
	3	2	1	×	×	×	×	×	×	×	×
	4	3	3	1	×	×	×	×	×	×	×
	5	4	6	4	1	×	×	×	×	×	×
	6	5	10	10	5	1	×	×	×	×	×
	7	6	15	20	15	6	1	×	×	×	×
	8	7	21	35	35	21	7	1	×	×	×
	9	8	28	56	70	56	28	8	1	×	×
	10	9	36	84	126	126	84	36	9	1	×
	11	10	45	120	210	252	210	120	45	10	1

Table 3.1 M values for various (N_S,N_G) pairs

Let us now consider how the code rate, R, varies with N_S and N_G. The size of each sub-frame as well as each $P\{\}$ sequence (Figure 3.4) is equal to the interleaver size, N_I, which was shown to be equal to $N_G.L_S$ earlier. As there are C_S sub-frames, the product of C_S, N_G, and L_S gives the total number of redundancy bits per information frame (with $N_S.L_S$ information bits). Therefore, without puncturing, R for a systematic multifold code can be expressed as in (3.3).

$$R = \frac{N_S \cdot L_S}{N_S \cdot L_S + C_S \cdot N_G \cdot L_S} = \frac{N_S}{N_S + C_S \cdot N_G} \qquad \textbf{(3.3)}$$

Corresponding code rates for N_S and N_G pairs listed in Table 3.1 are shown in Table 3.2. When the two tables are closely examined, it can be seen that systematic multifold codes with identical M values have the same code rates, which are inversely proportional to M. As a matter of fact, the exact relationship between R and M can be expressed as in (3.4). If the code is non-systematic, (3.4) simply becomes R=1/M.

$$R = \frac{1}{1+M} \qquad \textbf{(3.4)}$$

The validity of (3.4) can easily be proved as follows. Let us start by expanding C_S in equation (3.3) to obtain (3.5).

		N_G									
		2	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***
N_S	***2***	1/2	×	×	×	×	×	×	×	×	×
	3	1/3	1/2	×	×	×	×	×	×	×	×
	4	1/4	1/4	1/2	×	×	×	×	×	×	×
	5	1/5	1/7	1/5	1/2	×	×	×	×	×	×
	6	1/6	1/11	1/11	1/6	1/2	×	×	×	×	×
	7	1/7	1/16	1/21	1/16	1/7	1/2	×	×	×	×
	8	1/8	1/22	1/36	1/36	1/22	1/8	1/2	×	×	×
	9	1/9	1/29	1/57	1/71	1/57	1/29	1/9	1/2	×	×
	10	1/10	1/37	1/85	1/127	1/127	1/85	1/37	1/10	1/2	×
	11	1/11	1/46	1/121	1/211	1/253	1/211	1/121	1/46	1/11	1/2

Table 3.2 Unpunctured systematic code rates for various (N_S,N_G) pairs

Then, (3.5) can be simplified by cancelling out the like terms (see equation (3.6)). The right side of the denominator in (3.6) is equal to (3.7), which proves the validity of (3.4).

$$R = \frac{N_S}{N_S + \dfrac{N_S! \cdot N_G}{N_G! \cdot (N_S - N_G)!}} \qquad \textbf{(3.5)}$$

$$R = \frac{1}{1 + \dfrac{(N_S - 1)!}{(N_G - 1)! \cdot (N_S - N_G)!}} \qquad \textbf{(3.6)}$$

$$M = C(N_S\text{-}1, N_G\text{-}1) = \frac{(N_S - 1)!}{(N_G - 1)! \cdot (N_S - N_G)!} \qquad \textbf{(3.7)}$$

For the non-systematic case, the leftmost N_S term in the denominator in (3.5) disappears; therefore, R is equal to $1/M$.

3.3.3. Multifold Decoder

In the classical turbo decoding, the soft extrinsic information generated by one component decoder for an information frame, can be used as the a priori information for the other component decoder, either after interleaving or de-interleaving.

However, in multifold decoding, each component decoder processes one sub-frame, and therefore soft information at its input and output obeys the sub-frame format. As it was described in section 3.2, the encoded sub-frames have different segment combinations. For this reason, the soft information associated with each sub-frame should first be re-arranged before it can be utilised in iterative decoding. In this section, the general structure of multifold decoders will be described, and their operational and structural differences from the classical turbo decoders will be emphasised, where appropriate.

Multifold turbo decoder block diagram is presented in Figure 3.5. The decoder constitutes C_S component decoders, each of which processes a sub-frame C_a, and is denoted decoder(C_a). Note that $1 \leq a \leq C_S$. It should also be noted that the multifold component decoders are structurally identical to the ones used in classical turbo decoders.

There are C_S-1 interleavers and de-interleavers, denoted $\Pi(C_a)$ and $\Pi^{-1}(C_a)$. For C_1, no interleaving is necessary since I{C_1} is encoded without interleaving (see Figure 3.4). The major difference between a multifold decoder and the classical turbo decoder is the addition of multiplexer (mux) and demultiplexer (demux) blocks, which are used for reordering the soft extrinsic information in each sub-frame, during iterations. For each component decoder, there is one associated mux(C_a) and demux(C_a) block.

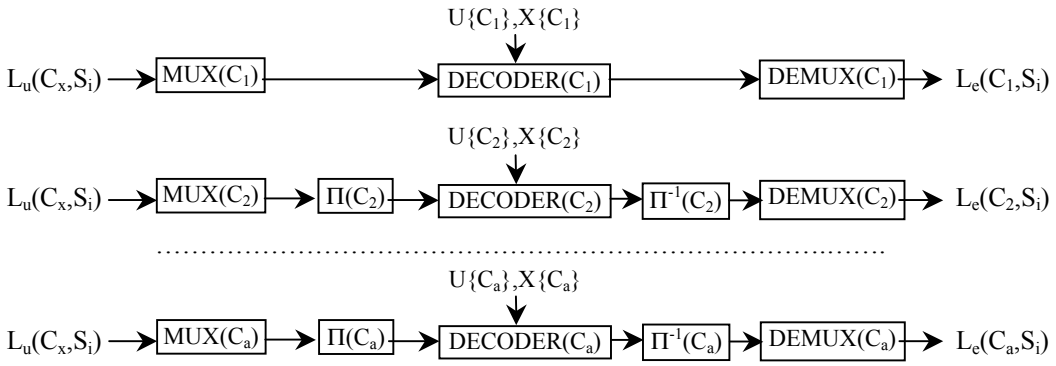


Figure 3.5 Multifold turbo decoder

Received information and parity symbols that are fed to decoder(C_a) are denoted as $U\{C_a\}$ and $P\{C_a\}$, respectively. Soft extrinsic information at the output of demux(C_a), i.e. $L_e(C_a,S_i)$, is associated with the information symbols in all segments S_i such that $\forall S_i$, $1 \le i \le N_S$, $S_i \in C_a$ is satisfied. The a priori information at the input of mux(C_a), i.e. $L_u(C_x,S_i)$, combines the demultiplexed L_e from C_S-1 component decoders such that $\forall S_i$, $1 \le i \le N_S$, $S_i \in C_x$, where $1 \le x \le C_S$ and $x \ne a$ is satisfied. The $x \ne a$ condition is necessary for performing accurate iterative decoding since none of the L_e values should be fed back to the decoder that generated it.

In classical turbo decoding, after the completion of iterations, a hard decision is made based on a posteriori soft values, L, in which the final L_e values and the channel information are combined. As for each information symbol, the multifold decoder generates M a posteriori values, the L value with the maximum magnitude is selected prior to hard decision (Figure 3.6).

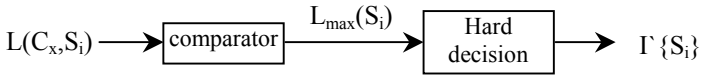


Figure 3.6 Multifold hard decision step

For a given i, such that $\forall C_x$, $1 \le x \le C_S$, and $S_i \in C_x$, M segments need to be compared in terms of the $L(C_x,S_i)$ values corresponding to their constituent symbols. The maximum soft value for each information symbol is selected out of M soft values, thus a set of L_{max} for each S_i, denoted $L_{max}(S_i)$, is available at the comparator output (Figure 3.6). As a final step, $L_{max}(S_i)$ soft value sequence is transformed into $\Gamma\{S_i\}$ binary array by the hard decision block.

3.4. Two-fold Turbo Code

As it was explained previously, N_S, the number of information segments in an information frame, and N_G, segments per sub-frame, are the two key parameters that define multifold turbo codes. When N_S and N_G parameters are set to 3 and 2, respectively, M becomes 2, which indicates that each information bit in a frame is encoded twice. This particular type of turbo code is referred to as the *two-fold turbo code*. Having described the general multifold encoding and decoding principles until now, a two-fold turbo code will be brought into focus in this section.

First, the two-fold encoding as well as parallel and serial decoding of two-fold turbo codes will be explained. Differences between the two-fold coding scheme and the equivalent

classical scheme will also be discussed. Then, the frame and bit error performance of two-fold codes are going to be presented for both decoding schemes. Finally, through a practical codeword weight analysis, reasons for the improved performance of the two-fold turbo codes will be given.

3.4.1. Two-fold Turbo Encoder

The two-fold encoder structure is shown in Figure 3.7. The information frame $I\{123\}$, consists of three segments, each with L_S bits. The segments, namely $I\{1\}$, $I\{2\}$, $I\{3\}$, are permuted in groups of two to generate three different sub-frames ($I\{12\}$, $I\{23\}$, $I\{13\}$). The length of each sub-frame determines the interleaving degree, i.e. N_I, which is equal to $2 \cdot L_S$. Two interleavers ($\Pi(23)$ and $\Pi(13)$) are necessary to provide statistical independence between the sub-frames, prior to encoding. Three parity codewords, each with length N_I, are generated ($P\{12\}$, $P\{23\}$, $P\{13\}$) after encoding. As the code is systematic, $I\{123\}$ is also a part of the output codeword.

In the classical encoder (Figure 3.8), two sets of parity bits $P_1\{1\}$, and $P_2\{1\}$ are generated from an information bit stream, $I\{1\}$, and its interleaved version, respectively. Here, the information frame can be considered both as a segment, and as a sub-frame, in multifold terms.

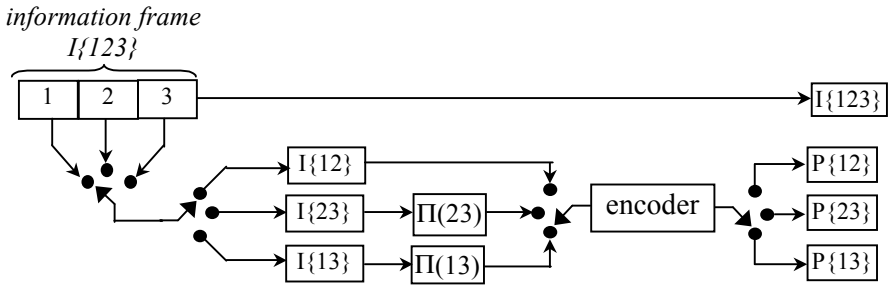


Figure 3.7 Two-fold turbo encoder

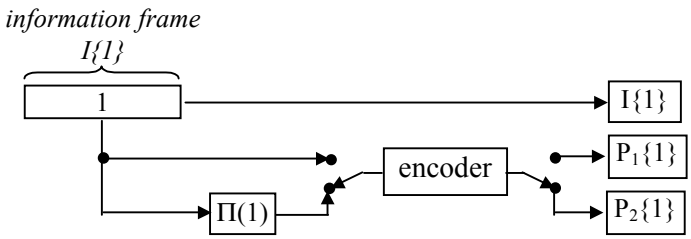


Figure 3.8 Classical turbo encoder

It was mentioned in section 3.3.2 that for systematic multifold codes, the mother code rate could be determined from (3.4) if M is known. Therefore, the mother code rate for two-fold turbo codes is 1/3.

3.4.2. Two-fold Turbo Decoder

In the two-fold serial decoder, (Figure 3.9), three identical component decoders process the three encoded sub-frames. The sub-frames in the two-fold case are indexed as 12, 13 and 23. Each component decoder has an associated mux and demux used for re-arranging relevant segments into sub-frames. It should be noted that the extrinsic information L_e at the output of each demux is shown as the a priori information at the input of the corresponding mux in the diagram.

After the completion of a fixed number of iterations, the final soft output of each constituent decoder is fed to a comparator. As there are two soft values available for each information symbol, the one with the highest magnitude is passed to the hard decision block. The hard decision block then assigns 1 and 0 hard values to the positive and negative soft values, respectively, to generate the decoded information frame, $\Gamma\{123\}$.

The classical serial decoder, on the other hand, comprises two component decoders (Figure 3.10). The first constituent component processes the received information and parity symbols, $U_1\{1\}$ and $X_1\{1\}$, while the second one accepts the interleaved version of the information symbols and the corresponding parity symbols, $U_2\{1\}$ and $X_2\{1\}$. Note that the L_e and L_u metrics are labeled similar to the two-fold decoder in (Figure 3.9). Hence, $L_e(1,1)$ indicates the L_e soft value that corresponds to the first segment of the sub-frame combination 1. As mentioned earlier, in classical turbo decoders there is one sub-frame and one segment, which correspond to the same information frame. Therefore, specifying the sub-frame and segment index is not necessary, however it provides consistency in our representation.

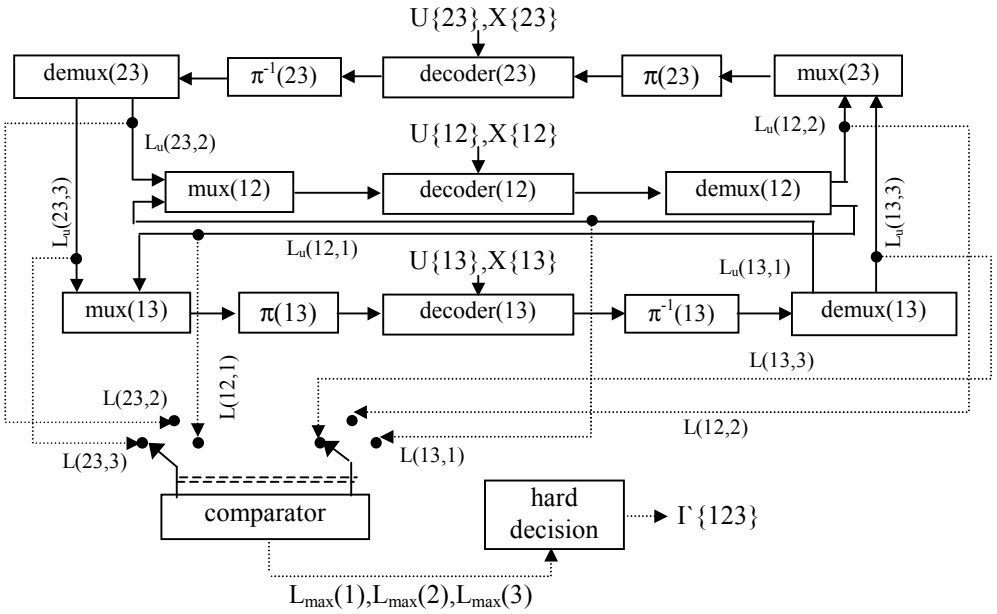


Figure 3.9 Two-fold turbo decoder – *serial operation*

Parallel decoder structures for the two-fold and the classical schemes are also presented in Figure 3.11 and Figure 3.12, respectively. Having explained the serial decoder structure in detail, only the differing aspects of parallel decoders will be discussed.

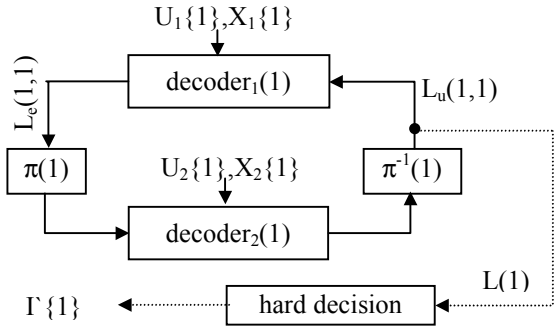


Figure 3.10 Classical turbo decoder – *serial operation*

It is important to note the primary difference between the serial and parallel turbo decoding. In the parallel operation the a priori information for each constituent decoder is initialized to zero before the first iteration. Therefore, each component decoder starts to operate with identical a priori log-likelihoods for each information symbol. However, in the serial mode depending on the component decoder that initiates the iterations, soft information generated by the consecutive components can be correlated with that decoder's extrinsic information. A typical example of this would be the classical serial decoder, where the second decoder a priori information is initialised by the first decoder's extrinsic output. Consequently, the extrinsic values generated by the second decoder will not completely be independent of the first decoder. Thus, the decoder is said to be biased towards the first component decoder. In other words, the soft value errors generated within the first component decoder degrade the performance of the second one. Due to this

operational difference, parallel decoding generates more reliable LLRs than the serial scheme. Advantages of parallel decoding were explained earlier in section 2.2.5.

The above disadvantage of serial operation affects the segments in two-fold decoding. That is to say, each information *segment* is biased towards a different decoder. This will be explained with an example.

Assume that iterations are started with decoder(12), which generates the first LLRs (Figure 3.9). Consequently, decoder(23) and (13) will initialize their operation with the available a priori values for segments 1 and 2. However, the a priori information for segment 3 will still be zero for those component decoders. If decoder(23) follows decoder(12), it is going to generate soft values for segments 2 and 3. Finally, decoder(13) will make use of a priori input for segments 1 and 3, which are biased towards two different component decoders, i.e. decoder(12) and (23).

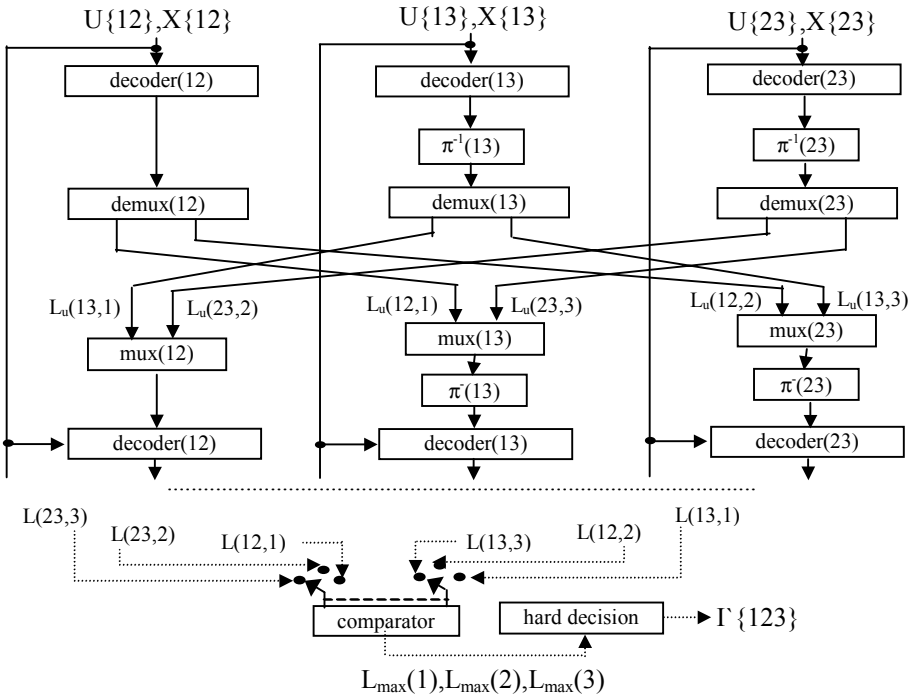


Figure 3.11 Two-fold turbo decoder – *parallel operation*

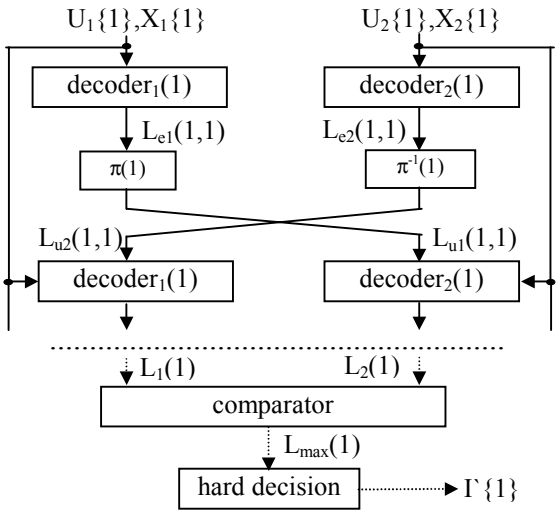


Figure 3.12 Classical turbo decoder – *parallel operation*

3.4.3. Two-fold Turbo Decoder Error Performance

Frame and bit error performance of the two-fold coding scheme has been evaluated using the max-log MAP algorithm for two different codes, namely the RSC g(7,5) and the CCSDS recommendation code, RSC g(23,33). In order to provide a fair comparison, each code has been compared to its equivalent classical turbo-coding scheme. As the codes were not punctured, the overall rate was fixed at 1/3 for all coding schemes. The information frame size was chosen as 4608 bits, and the error performance was assessed for serial and parallel decoding configurations.

First, the error performance results for serial decoding are presented. The bit error performance of the RSC g(7,5) two-fold code and its classical equivalent is compared in Figure 3.13 for iterations 2, 4, 8 and 16. In the plots, '*C*' and '*T*' stand for the *classical* and the *two-fold* cases, respectively.

For 2 iterations, the two-fold scheme is observed to perform worse than its classical equivalent, however, for 4 iterations and above, the two-fold scheme performs better in the waterfall region. In fact, after 1.5 dB, 4 two-fold iterations are enough to outperform 16 classical iterations. The frame error performance also shows a similar trend as the bit error performance for this code (see Figure 3.14).

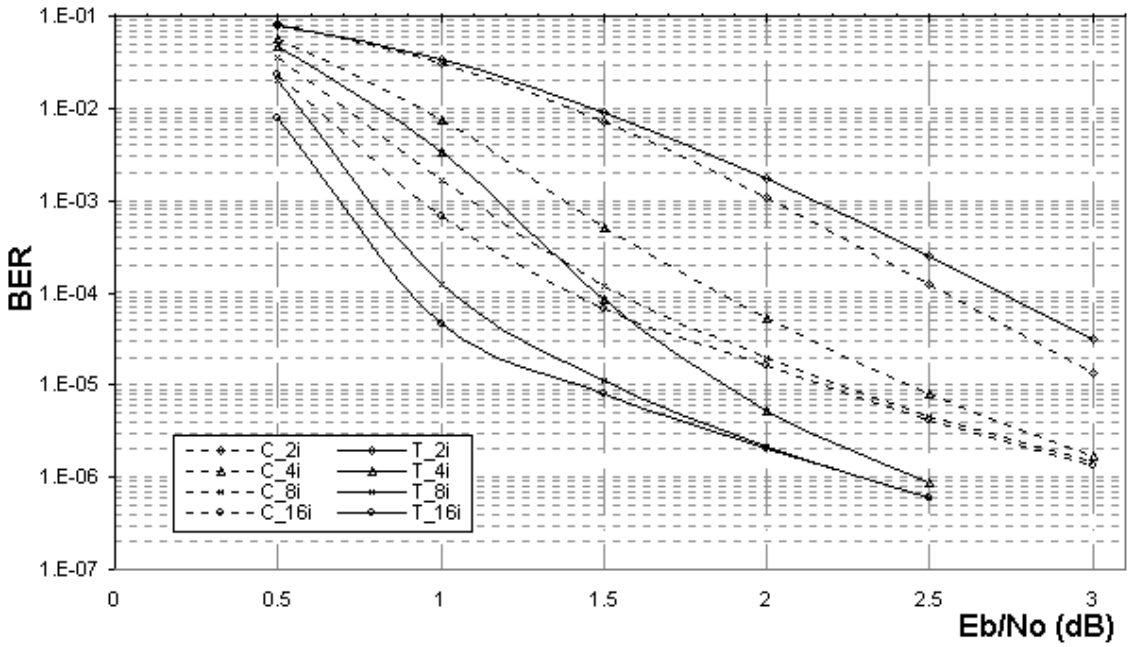


Figure 3.13 Two-fold bit error performance of the g(7,5) code – *serial decoding*

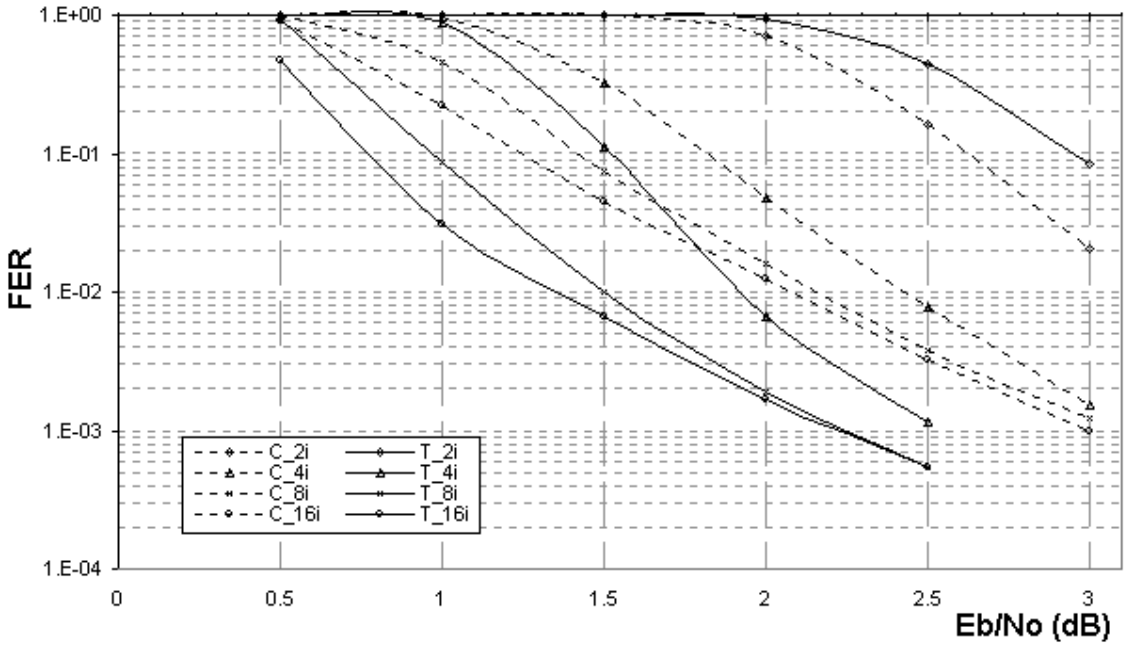


Figure 3.14 Two-fold frame error performance of the g(7,5) code – *serial decoding*

When the BER of the g(23,33) code is considered, the two-fold scheme is observed to outperform the classical one for 4 and 8 iterations (Figure 3.15). For 2 and 16 iterations, the two-fold bit error performance is worse than the classical one. However, the frame error results presented in Figure 3.16 unveils an advantage of the two-fold turbo codes. Even though, the classical turbo decoder corrects more bit errors than the two-fold decoder with 16 iterations, the two-fold decoder has a lower frame error rate.

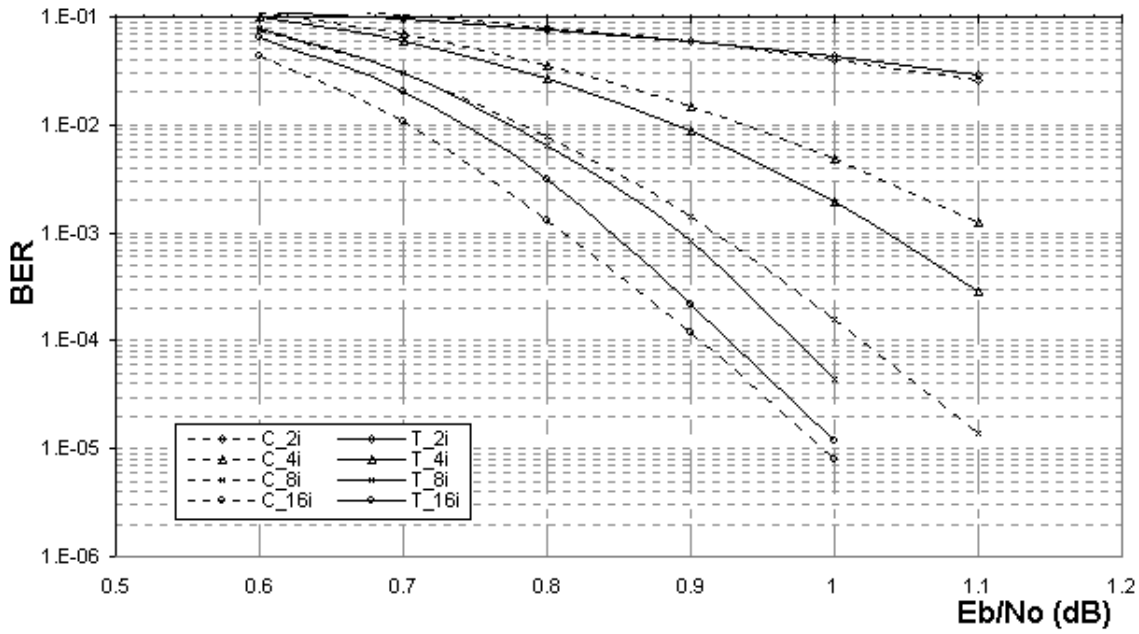


Figure 3.15 Two-fold bit error performance of the g(23,33) code – *serial decoding*

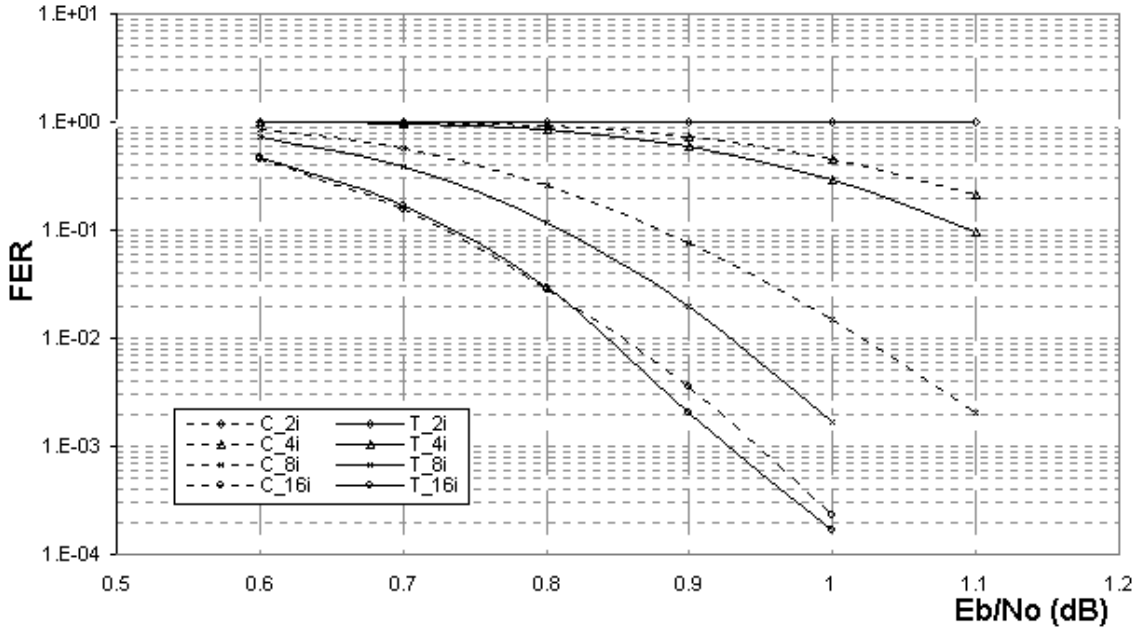


Figure 3.16 Two-fold frame error performance of the g(23,33) code – *serial decoding*

Due to the nature of serial turbo decoding, the LLRs of the component decoders are generally biased towards the first component. This is because during the first iteration, only the first component decoder generates soft extrinsic value based on zero a priori input, in which case the only decoding error source is the noise in the channel. However, the superseding component decoders generate soft outputs that are affected by two sources of error; the channel noise and the erroneous a priori information generated by the preceding component decoders. Therefore, in serial decoding the first component decoder generally has an advantage over the others, as it is able to generate unbiased soft output. A few

researchers have also addressed the relationship between the decoding configurations and the error performance previously [Han & Takeshita, 01] .

In serial two-fold turbo decoding, the error performance is closely associated with the operation order of the component decoders. As there are three component decoders in the two-fold turbo decoder (Figure 3.9), 3! serial decoding configurations are possible, which are indicated in Table 3.3.

To show the effect of using different decoding configurations on the error performance, all possible two-fold serial decoding patterns were evaluated for the g(7,5) and the g(23,33) codes, using 4 iterations.

decoding configuration	***first component decoder***	***second component decoder***	***third component decoder***
1	decoder(12)	decoder(13)	decoder(23)
2	decoder(12)	decoder(23)	decoder(13)
3	decoder(13)	decoder(12)	decoder(23)
4	decoder(13)	decoder(23)	decoder(12)
5	decoder(23)	decoder(12)	decoder(13)
6	decoder(23)	decoder(13)	decoder(12)

Table 3.3 Two-fold serial decoding configurations

For the g(7,5) code, the best BER and FER performance was achieved for the second configuration in Table 3.3, while the fourth configuration performed the worst (Figure 3.17 and Figure 3.18). In the case of the g(23,33) code, the frame and bit error performance ranking was observed to stay the same (Figure 3.19 and Figure 3.20). The fifth decoding configuration was found to perform somewhere between the best and the worst serial configurations, and therefore this configuration was preferred for evaluating the two-fold serial decoding error performance presented until now.

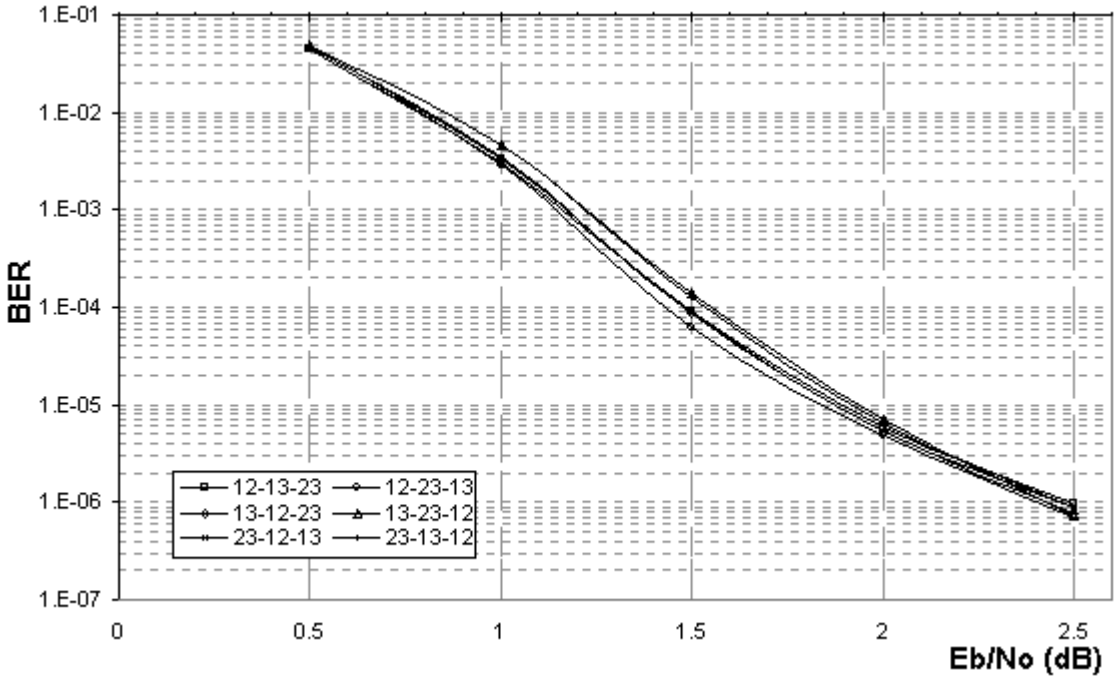


Figure 3.17 Two-fold bit error performance of the g(7,5) code for different serial decoding configurations

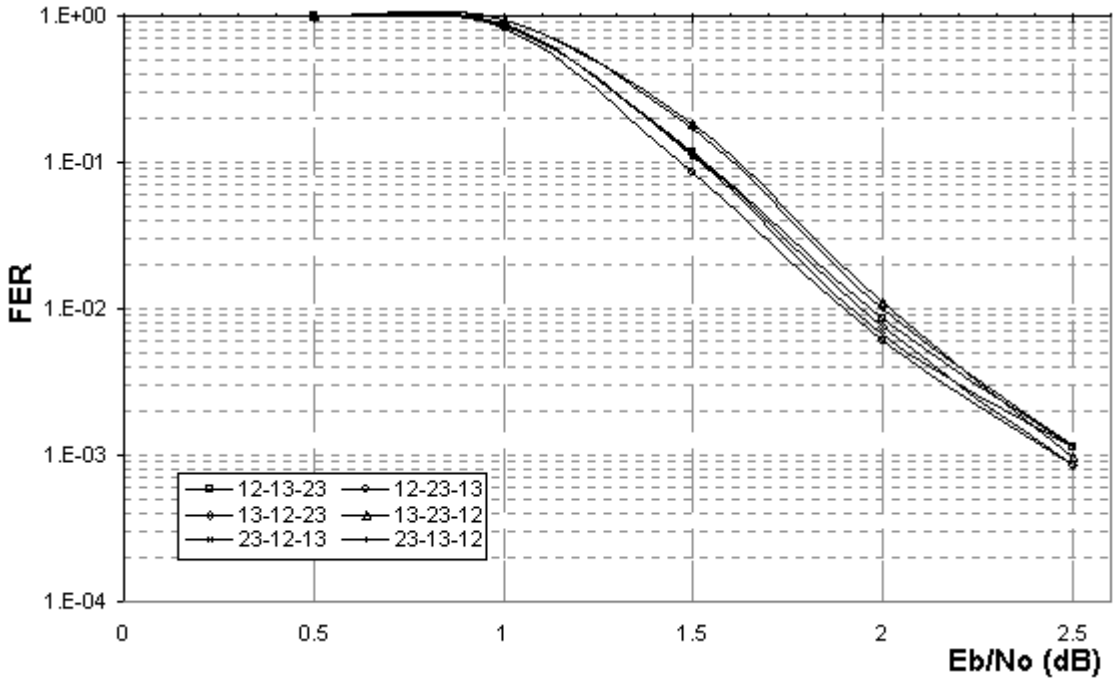


Figure 3.18 Two-fold frame error performance of the g(7,5) code for different serial decoding configurations

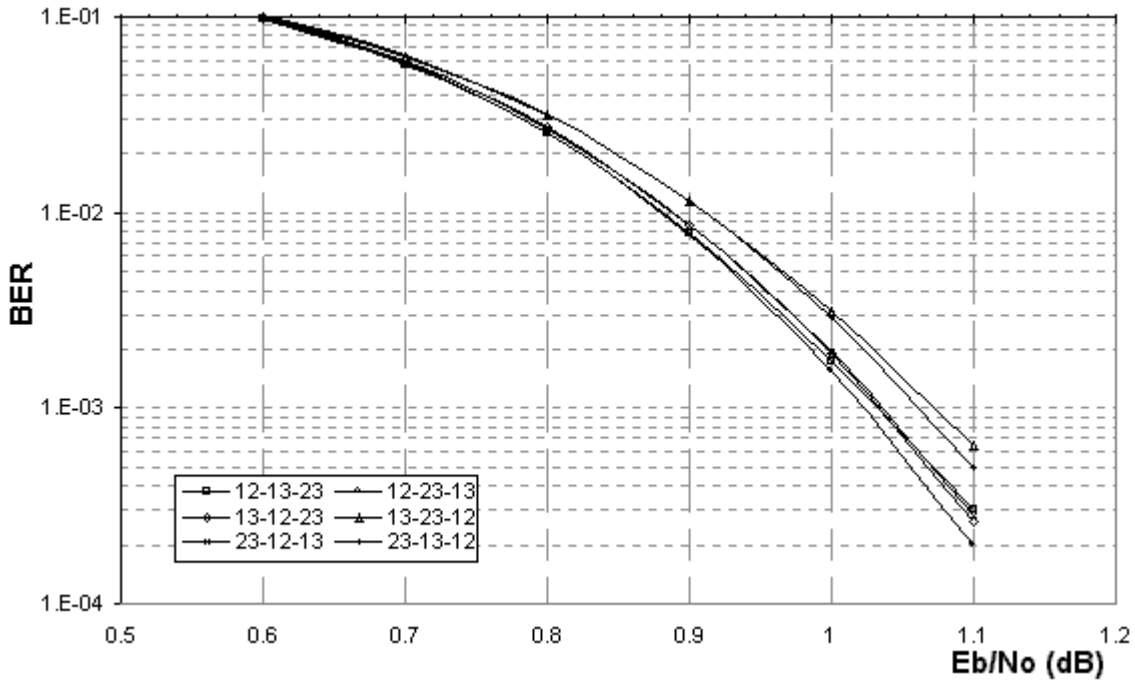


Figure 3.19 Two-fold bit error performance of the g(23,33) code for different serial decoding configurations

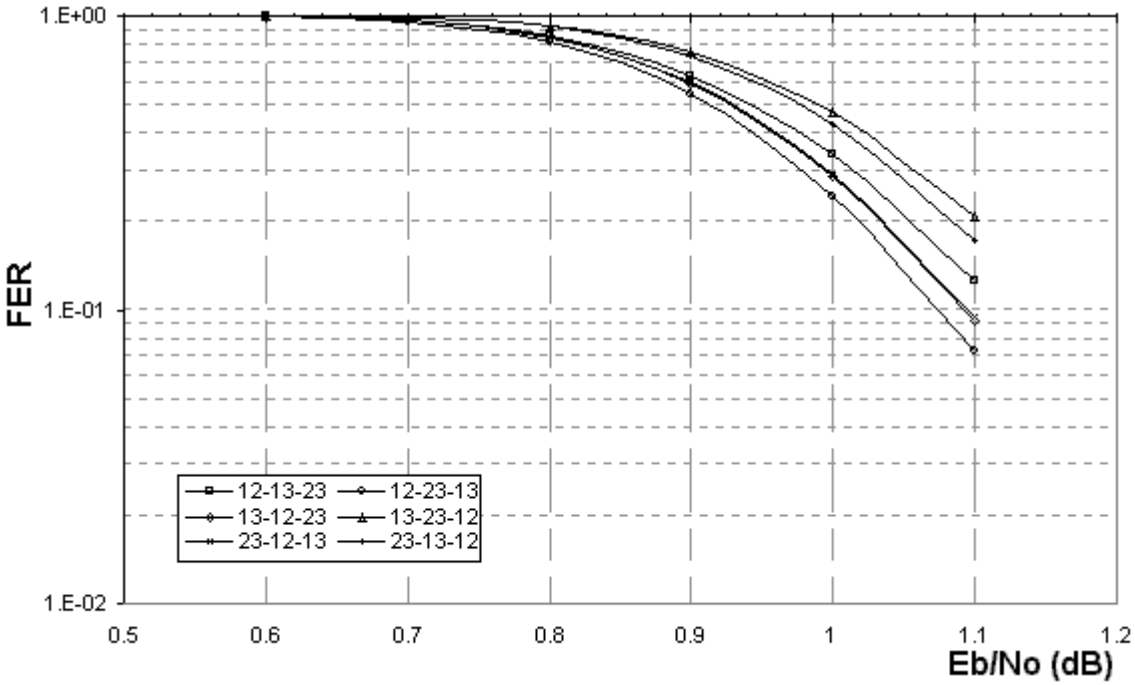


Figure 3.20 Two-fold frame error performance of the g(23,33) code for different serial decoding configurations

The error performance has also been evaluated for parallel decoding configurations for the g(7,5) and the g(23,33) codes.

When the bit error of the g(7,5) code is considered for parallel decoding, the superior performance of the two-fold turbo code can be observed even at 2 iterations (Figure 3.21). Two iterations using two-fold decoder performs better than 4 classical iterations at all times, and it outperforms 8 and 16 classical iterations after 1.25 and 1.75 dB, respectively. Also the frame error performance of the two-fold turbo decoder is far better than that of the classical's (Figure 3.22).

The improved bit and frame error rates are also confirmed by the parallel decoded g(23,33) two-fold turbo code. Up to 16 iterations, the bit error performance of the two-fold scheme is better than the classical turbo decoder's (Figure 3.23). The bit error performance of 4 two-fold iterations is identical to that of 8 classical iterations at 1.0 dB. For 16 iterations, the classical turbo decoder outperforms the two-fold decoder by approximately 0.06 dB at BER=10^{-5}. However, the frame error performance of the two-fold decoder compensates for the coding loss observed in bit error performance (Figure 3.24). In fact, the FER of 4 two-fold iterations is better than that of the 8 classical iterations even at signal-to-noise ratios less than 1.0 dB where the BER of the two-fold decoder is worse than the classical's.

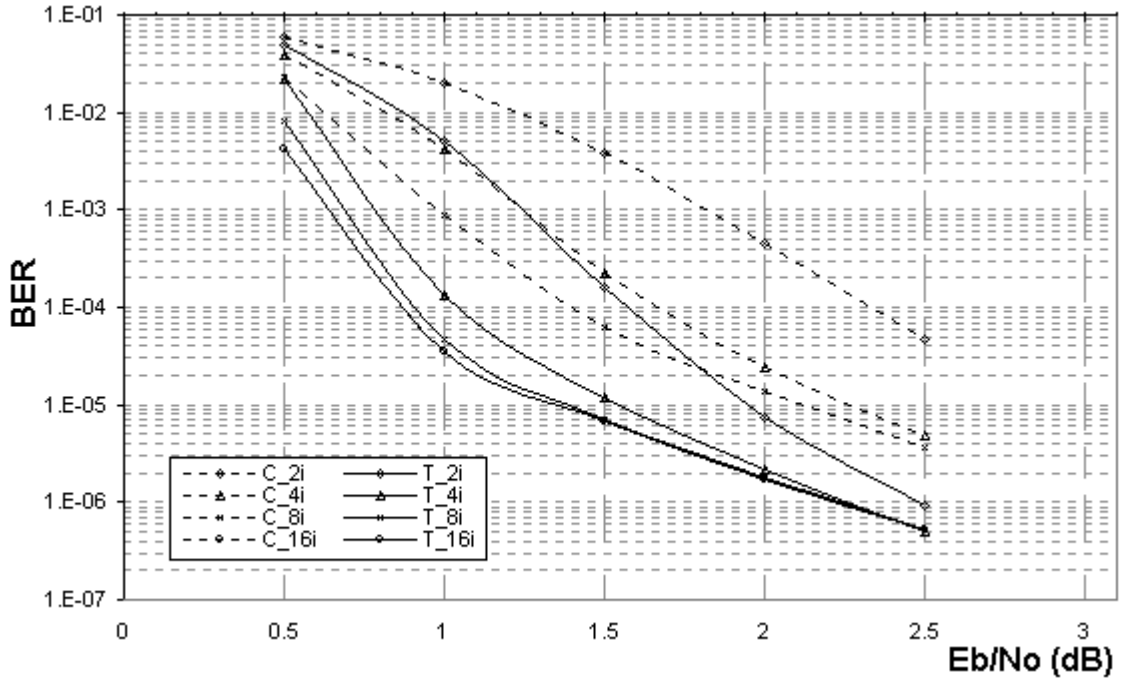


Figure 3.21 Two-fold bit error performance of the g(7,5) code – *parallel decoding*

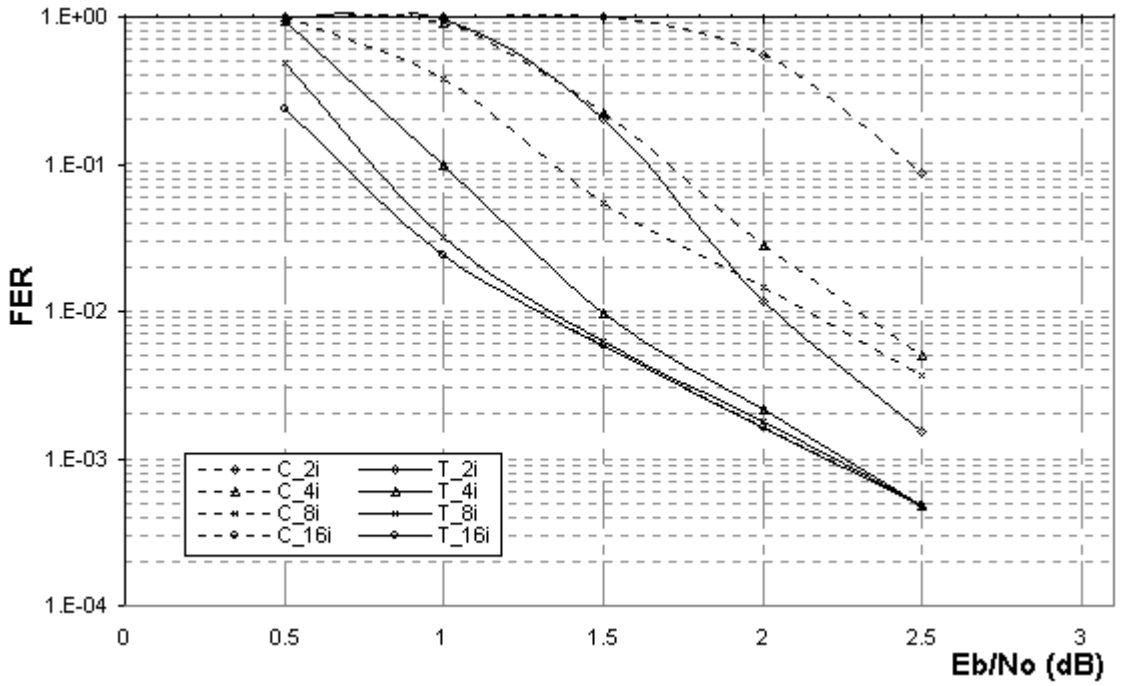


Figure 3.22 Two-fold frame error performance of the g(7,5) code – *parallel decoding*

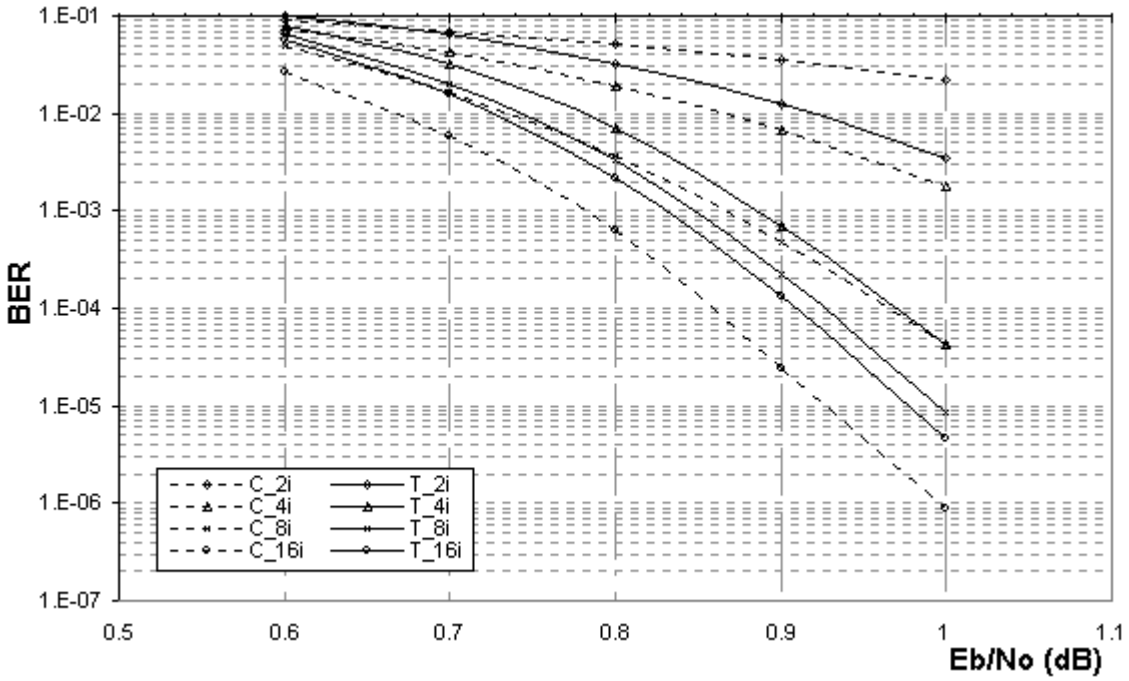


Figure 3.23 Two-fold bit error performance of the g(23,33) code – *parallel decoding*

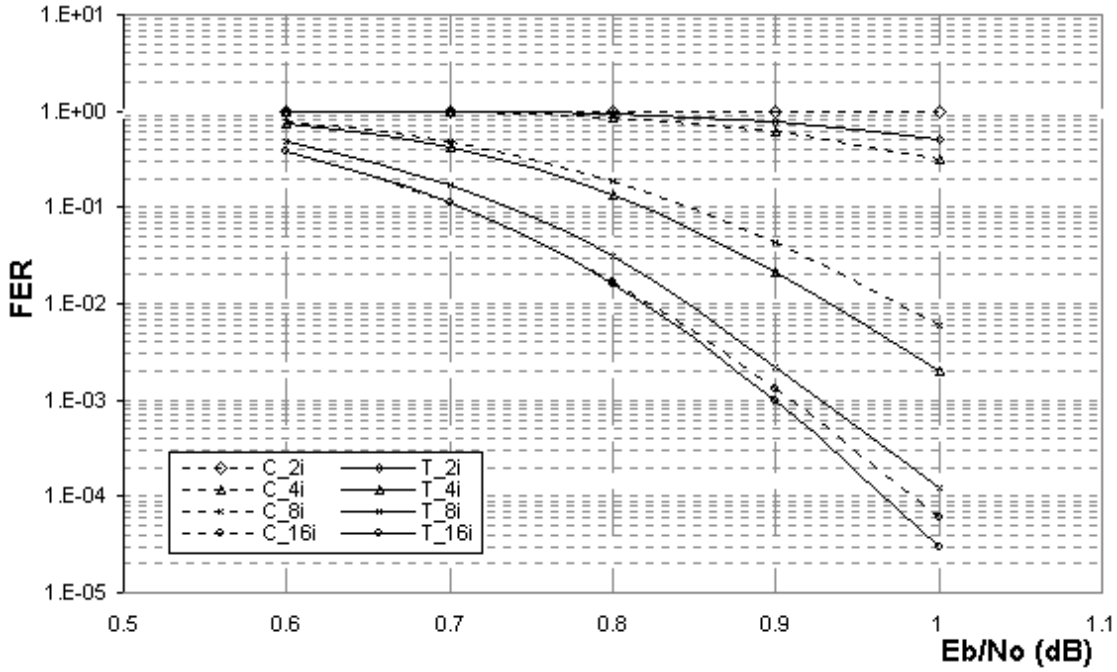


Figure 3.24 Two-fold frame error performance of the g(23,33) code – *parallel decoding*

3.4.4. Two-fold Codeword Weight Distribution

The coding gain discussed in the previous section, is closely related to the improved codeword weight distribution properties achieved by the two-fold turbo coding. In literature, the randomness of the codeword weight distribution has been shown to be a reliable indicator of code performance, besides good distance properties.

Svirid [Svirid, 95] has mentioned that the weight distribution of the terminated convolutional codes could be approximated by a Gaussian distribution for feed-forward and feedback encoders. He has also shown that the codeword weight distribution could be close to random when feedback convolutional encoders were used.

Battail [Battail, 98] has also suggested that good long linear codes mimic the weight distribution of random coding rather than having a large minimum distance. He has exemplified this by referring to the iterated product of single-parity-check (SPC) codes, which may be seen as the precursor of turbo codes. Despite their poor minimum distance properties, the SPC codes perform well because of their random weight distribution, which was shown by Biglieri and Volski [Biglieri & Volski, 94] .

Battail has also presented significant insights regarding one key ingredient of turbo coding, namely interleaving. From a weight distribution perspective, a secondary function of interleaving can be described as shaping the codeword weight distribution. Besides

minimizing the correlation between the turbo encoded information sequences, interleavers also make the weight distribution more random, which has also been identified by Svirid [Svirid, 95] .

It is not wrong to say that the bit error performance of turbo codes partly depends on how random the codeword weight distribution is. The multifold turbo codes perform better than their equivalent turbo codes primarily because they bring the weight distribution a step closer to that of perfect random [Tanriover et al, 01a] .

In this section, we are going to demonstrate how random the weight distributions for the two-fold turbo codes are compared to those of their equivalent classical turbo schemes. The approach presented in this section is purely based on simulation data and is by no means conclusive. However, it gives important clues to why the two-fold turbo codes perform so well.

The weight distribution analysis was performed for the g(7,5) RSC as well as the CCSDS recommendation code, g(23,33), for R=1/3 with codeword length 13824 bits (excluding the termination bits). For each code, two simulations were conducted; one for the classical turbo encoding and another for the two-fold equivalent. During each simulation, 25 million codewords were generated, and the cumulative weight distribution for each scheme was analyzed. The input information bits were generated randomly, and identical information frames were encoded in each coding scheme to allow fair comparison.

The cumulative weight distributions revealed that certain weight were not generated by any of the 25 million information frames, at all. In other words, there were *weight gaps* in the distribution, which would not occur if the codeword weights were randomly distributed. Intuitively, one would expect the number of such gaps to decrease, as the weight distribution approached ideal randomness.

The weight gap distribution for the g(7,5) and the g(23,33) RSC codes are shown in Figure 3.25 and Figure 3.26, respectively, for the two-fold and the classical turbo coding schemes. The dashed lines show the range of weights that are generated for each scheme. Each circle or square indicates a gap in the corresponding weight value on the horizontal axis. For perfect random distribution, no gaps within the given weight range are observed.

As can be seen from Figure 3.25 and Figure 3.26, the two-fold turbo encoder generates higher weight codewords than its classical equivalent.

Another difference between the two distributions is their *gap ratio* (*GR*), which is defined as in (3.8), where N_w denotes the number of weights within the valid weight range, and N_g is the number of gaps within that range. The *GR* can be used to enumerate the randomness of the codeword weight distribution; as *GR* gets smaller, the distribution is said to approach to perfect random.

$$GR = \frac{N_g}{N_w} \cdot 100 \qquad \textbf{(3.8)}$$

For the g(7,5) code, the *GR* for the two-fold and the classical scheme are calculated as 47.68% and 52.70%, respectively – a result that suggests the weight distribution of the two-fold scheme is approximately 5% closer to random distribution than its classical equivalent.

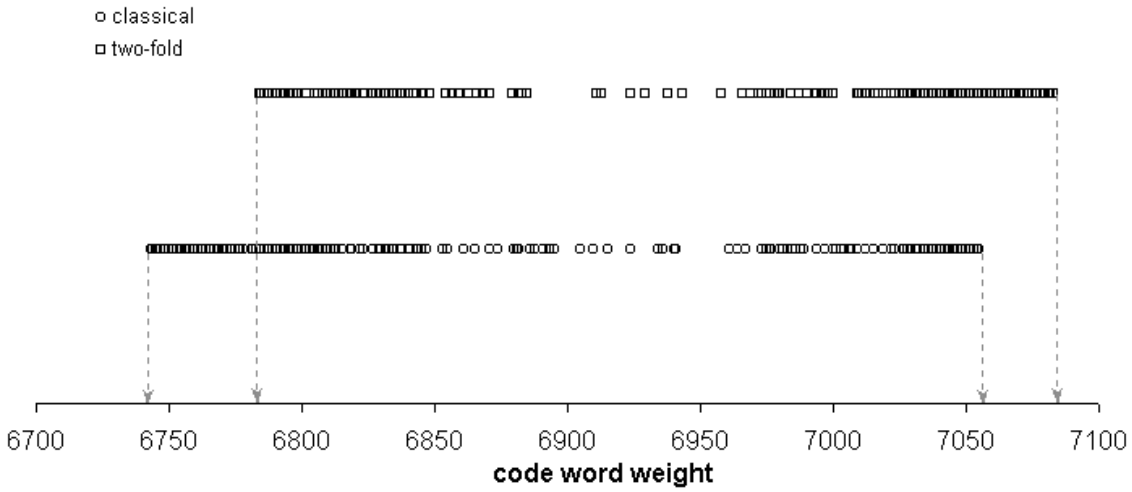


Figure 3.25 Codeword weight gap distribution for the g(7,5) RSC code

A similar result was obtained in the case of the g(23,33) RSC code; the *GR* for the two-fold and the classical scheme are found to be 55.83% and 57.95%, respectively, which again implies that the two-fold coding distributes the codewords more randomly than the classical scheme.

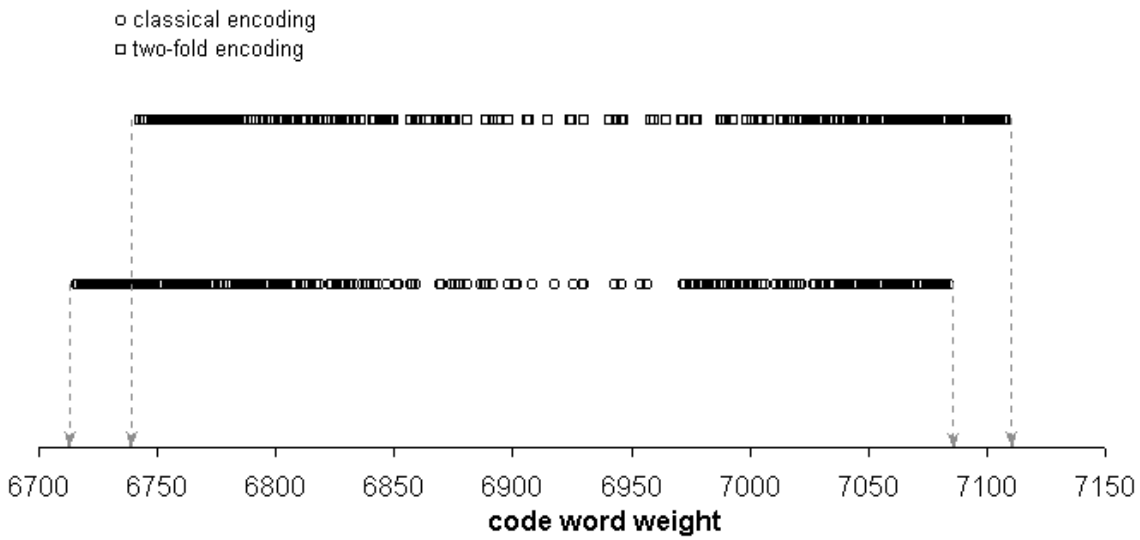


Figure 3.26 Codeword weight gap distribution for the g(23,33) RSC code

3.4.5. Two-fold Decoder Complexity

In this section, processing time is used as a measure to quantify the decoder complexity. The overall decoding time is calculated by adding the processing time of the constituent blocks that have the highest computational shares.

First, a general expression that represents the complexity of a decoder sub-block needs to be introduced. The processing time spent by a constituent block, *x*, will be represented by T_x, and will be calculated as in (3.9) in its general form. In this expression, N_{op} is the number of operations performed in block *x*, and t_{op} stands for the processing time spent per operation in the same block. In the rest of this section, representation in (3.9) is going to be used as a reference for each block.

$$T_x = N_{op} \cdot t_{op} \qquad \textbf{(3.9)}$$

In the scope of the analysis explained here, 7 blocks within the decoders are taken into account. They are the component decoder (*cd*), interleaver (Π), deinterleaver (Π^{-1}), multiplexer (*mx*), demultiplexer (*dmx*), comparator (*cmp*) and hard decision (*hd*) blocks. The *mx* and *dmx* blocks are assumed to have a similar degree of complexity as they perform the inverse operations of each other. Same argument is also valid for Π and Π^{-1} blocks. Therefore, the complexity evaluation of 5 blocks is sufficient to obtain a figure for the overall decoding complexity. Complexity assessment for each of those blocks will be explained next.

The component decoder complexity is directly proportional to the decoding trellis size and the decoding algorithm used. In our application the sub optimal max-log-MAP decoding algorithm is used.

The trellis decoding operations refer to floating point operations (fpos), namely additions and multiplications performed at all states and depths. The number of states, S, is $2^{(K-1)}$, where K is the constraint length of the convolutional code. The interleaver length, N_I, gives the number of trellis depths, excluding the tail section that resets the encoder to zero state. For large interleaver lengths (≥4000) operations at the tail of the trellis can be ignored, as it does not introduce a relatively large computational overhead. Also note that for the conventional and the multifold schemes, N_I is equal to N_B and $N_G \cdot L_S$, respectively. As in the binary decoding trellis (Figure 2.15) there are 2 bits per branch, and therefore for each depth, 4 branch probability (gamma) calculations are performed. Since each gamma value is obtained after 4 fpos, a total of 16 fpos per depth are necessary. Every state in the trellis is assigned an alpha and a beta state probability, each of which requires 2 fpos, giving a total of 4 fpos per trellis state. Assume that trellis decoding is executed by a microprocessor that carries out one fpo in t_{fpo} seconds. Then, for one component decoder, processing time per iteration (T_{cd}) can be generalized as in (3.10).

$$T_{cd} = \left(16 \cdot N_I + 2^{K+1}\right) \cdot t_{fpo} \qquad \textbf{(3.10)}$$

Interleaving time depends on the interleaving degree, which is determined by the number of symbols in an information frame. If it takes $t_{i/d}$ seconds to interleave (or de-interleave) one symbol, then the overall interleaving time for one N_I, $T_{i/d}$, can be represented as in (3.11).

$$T_{i/d} = N_I \cdot t_{i/d} \qquad \textbf{(3.11)}$$

The *mx* and *dmx* blocks do not exist in the classical turbo decoder. In a multifold decoder, one *mx* (or *dmx*) block parses $N_G \cdot L_S$ symbols. For $t_{mx/dmx}$ seconds parsing time per symbol, the processing time spent by one *mx* or *dmx* block on one sub-frame, $T_{mx/dmx}$, is given by (3.12).

$$T_{mx/dmx} = N_G \cdot L_S \cdot t_{mx/dmx} \qquad \textbf{(3.12)}$$

The *cmp* block, compares the LLRs in relevant segments to maximise the decoding confidence, prior to hard decision. In multifold decoders, for each information symbol, (M-1) such comparisons are performed. Given that one such comparison takes t_{cmp} seconds,

and that a total of $(M\text{-}1)\cdot N_S\cdot L_S$, then the total time spent by *cmp* per information frame can be calculated from (3.13).

$$T_{cmp} = (M-1)\cdot N_S \cdot L_S \cdot t_{cmp} \qquad \textbf{(3.13)}$$

The *hd* block in both the conventional and multifold case, processes $N_S\cdot L_S$ (or N_B) symbols for each information frame. Assume that it takes t_{hd} seconds for a processor to apply hard decision to one decoded symbol. In this case the *hd* block requires T_{hd} seconds to complete its operation as given by (3.14).

$$T_{hd} = N_S \cdot L_S \cdot t_{hd} \qquad \textbf{(3.14)}$$

Number of the above blocks, vary depending on the turbo decoder used. Therefore, in order to calculate the overall complexity for the classical and the multifold decoders, the total number of blocks needs to be generalised.

Table 3.4 summarizes the complexity in terms of the number of operations performed. The first column in the table lists the name of the sub-blocks that contribute significantly to the overall turbo decoder complexity. The left half of the remaining four columns belongs to the classical decoder. The second column of the table lists the number of relevant sub-blocks used in the classical decoder structure. The total number of operations performed by each sub-block is displayed in column three of the same table. The same information is also listed for the multifold turbo decoder in the last two columns, in the same order.

	Classical Turbo Decoder		***Multifold Turbo Decoder***	
sub-block name	**total number of blocks**	**total number of operations**	**total number of blocks**	**total number of operations**
cd	2	$2\cdot(16\ N_B+2^{K+1})$	$C(N_S,N_G)$	$C(N_S,N_G)\cdot(16\ N_G\cdot L_S+2^{K+1})$
Π and Π^{-1}	2	$2\cdot N_B$	$2\cdot[C(N_S,N_G)\text{-}1]$	$2.[C(N_S,N_G)\text{-}1]\cdot N_G\cdot L_S$
mx and dmx			$2\cdot C(N_S,N_G)$	$2\cdot C(N_S,N_G)\cdot N_G\cdot L_S$
cmp			1	$(M\text{-}1)\cdot N_S\cdot L_S$
hd	1	N_B	1	$N_S\cdot L_S$

Table 3.4 Number of operations for classical and multifold decoders – *serial decoding*

In order to convert Table 3.4 into time format, all the t_{op} values presented in (3.10) through (3.14), need to be incorporated. Therefore, the t_{op} parameters were first measured using stopwatch functions, which are available as libraries in standard C programming language, and provide up to 1 microsecond accuracy. Each t_{op} measurement was repeated 100 times to validate the consistency of the measurements. In order to express the complexity analysis independent from the type of processors used, measurements are presented relative to one '*and*' operation, which introduces a processing delay of T seconds. Table

3.5 presents the processing delays in units of T, for the decoder sub-blocks discussed so far.

The last step in the complexity evaluation of multifold decoders is combining Table 3.4 and Table 3.5. This is simply done by multiplying corresponding rows of the total number of operations with the normalised operation delays. Note that for the classical turbo decoders *mx/dmx* and *cmp* blocks are ruled out in Table 3.4.

Operation	Normalised Delay
t_{fpo}	$7.1T$
$t_{i/d}$	$2.9T$
$t_{mx/dmx}$	$1.6T$
t_{cmp}	$18.5T$
t_{hd}	$5.7T$

Table 3.5 Normalised operation delays

Table 3.6 presents the total time delay (or operation time) of each sub-block in units of T. Sum of all the expressions in a column, gives the overall complexity for the corresponding turbo decoder. By using this table, the complexity of any multifold decoder (for serial decoding) can easily be compared to an equivalent classical turbo decoder's.

sub-block name	Total sub-block operation time in a *classical turbo decoder*	Total sub-block operation time in a *multifold turbo decoder*
cd	$2 \cdot (16\ N_B + 2^{K+1}) \cdot (7.1T)$	$C(N_S,N_G) \cdot (16\ N_G \cdot L_S + 2^{K+1})\ (7.1T)$
Π and Π^{-1}	$2 \cdot N_B \cdot (2.9T)$	$\{2.[C(N_S,N_G)\text{-}1] \cdot N_G \cdot L_S\}\ (2.9T)$
mx and dmx		$\{2 \cdot C(N_S,N_G) \cdot N_G \cdot L_S\}\ (1.6T)$
cmp		$[(M\text{-}1) \cdot N_S \cdot L_S]\ (18.5T)$
hd	$N_B \cdot (5.7T)$	$N_S \cdot L_S\ (5.7T)$

Table 3.6 Number of operations for classical and multifold decoders – *serial decoding*

If we denote the overall complexity of the classical and the two-fold turbo decoders as C_{ctd} and C_{ttd}, respectively, the computational overhead introduced by the two-fold scheme (O_{ttd}) can be calculated from (3.15).

$$O_{ttd} = \frac{C_{ttd} - C_{ctd}}{C_{ctd}} \cdot 100 \qquad \textbf{(3.15)}$$

As an example, let us consider the two-fold turbo decoder and its classical equivalent. The parameters required for the complexity evaluation are as follows:

K=3	N_S=3	M=$C(N_S$-1, N_G-1)=2
N_B=4608	N_G=2	L_S=N_B/N_S=1536

Considering the above parameters, the two-fold decoding delay is calculated to be *8.67% higher* than its classical equivalent. Also in the case of K=5, this figure was found to be the same.

A closer examination of Table 3.6 reveals a couple of important points about the complexity of multifold turbo decoding.

The first point is that the primary portion of the computational overhead introduced by the multifold decoders, operating in series, comes from the multiplexing and de-multiplexing operations as well as the comparisons prior to hard decision. More importantly, such overheads increase with the fold of the turbo decoder.

The second point is related to the component decoder complexity in multifold codes. As the fold of the turbo codes increases, the overhead introduced by the multifold component decoders becomes more pronounced. Figure 3.27 illustrates this argument. The minimum *cd* overhead is introduced in the case of two-fold turbo decoders, which is about 0.01%. However, as the fold approaches 10, the overhead moves towards 400%. It can also be seen in Figure 3.27 that the *cd* complexity overhead characteristic is almost identical for turbo decoders with constraint lengths 3 and 5.

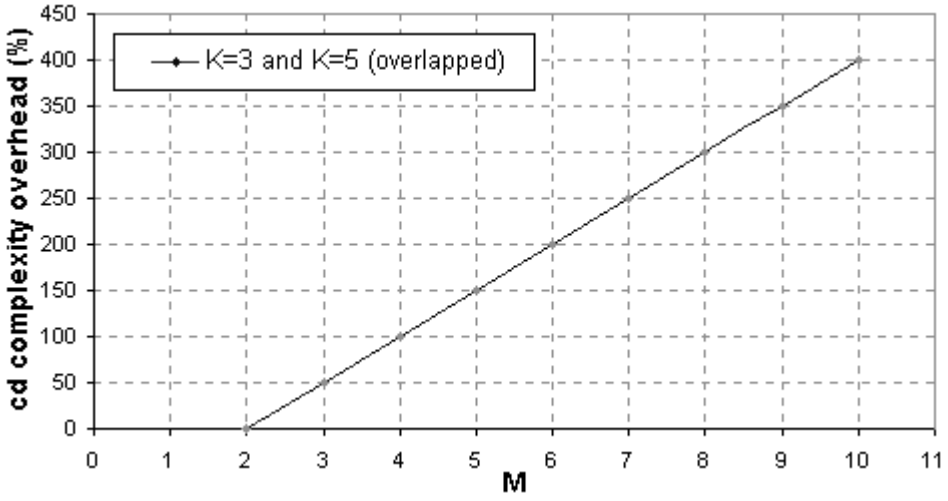


Figure 3.27 Component decoder (*cd*) complexity overhead versus fold parameter, M

3.5. Discussion

This chapter has introduced the multifold turbo coding, which improves the error performance of traditional turbo codes in the waterfall region. Multifold encoding and decoding have been explained in detail, and the differences from the classical turbo coding have been emphasised where appropriate. As an example, the two-fold turbo coding has been evaluated in terms of its error performance, weight distribution and decoding complexity, using the g(7,5) and the g(23,33) RSC codes.

For the g(7,5) RSC code, performance improvement up to 0.6 dB were achieved with the parallel and serial decoding. The simulation results for serial decoding have shown that 4 two-fold iterations can achieve better performance than 16 classical iterations for moderate to high signal-to-noise ratios. When two-fold codes are decoded in parallel, 2 iterations were found to perform better than 4 and, in some cases, 8 classical iterations under improved channel conditions.

The improved error performance of the two-fold turbo codes were also observed for the g(23,33) RSC code. With this code, the highest coding gain was achieved with parallel decoding, which was approximately 0.15 dB. Another important advantage of the two-fold turbo codes in terms of frame error performance was observed for 16 iterations. Even though, the two-fold decoder's BER was worse than that of the classical decoder's, its frame error performance was better for both serial and parallel decoding. In fact, for parallel decoding, the FER of 4 two-fold iterations was even better than 8 classical iterations.

The multifold turbo coding increases the randomness of the codeword distribution, which improves the error performance in the waterfall region. For the two-fold turbo code, the weight distribution can be made 2 to 5% more random for the g(23,33) and the g(7,5) RSC codes, respectively.

It is important to note that using multifold turbo codes increases the cost of decoding in terms of computations. The two-fold turbo codes are the least complex members of the multifold family, and the decoder complexity increases when the fold, i.e. M, is increased. Analysis has shown that the two-fold decoders increase the classical turbo decoding latency by approximately 8.67%, which was found to be roughly the same for the g(7,5) and the g(23,33) codes.

Chapter 4

Iteration Control with Soft Information

4. Iteration Control with Soft Information

4.1. Introduction

Increasing the number of iterations improves the bit error performance of turbo decoders at the expense of increased power consumption and decoding delay. Even though convergence to the maximum likelihood solution for most information bits is achieved by performing more iterations, the dynamic nature of the turbo decoder does not ensure convergence for all bits. Besides quick convergence at early iteration steps, oscillations and unstable a posteriori information progress patterns are also observed at the turbo decoder. For such patterns, increasing the number of iterations brings nothing more than computational overhead.

In order to increase the iterative decoding efficiency in terms of time and processing, researchers in the field of error coding have proposed a number of error detection and stopping criteria suitable for turbo codes [Shibutani et al, 99] , [Reid et al, 01] , [Zhai & Fair, 01] .

One interesting work by Buckley and Wicker is a neural network scheme that processes the cross entropy of the turbo decoder for detecting bit errors within a frame [Buckley & Wicker, 00] , [Buckley & Wicker, 99] . They have developed “Future Error Detecting Network” (FEDN), which minimizes the “wasted” number of iterations and “Decoder Error Detecting Network” (DEDN), which increases the reliability by predicting the existence of errors within a frame. The authors have reported that their technique starts failing when the number of errors within a frame gets small - a situation which can easily be compensated by adding a CRC to each frame. Also, the network needs to be trained by using a sufficiently high number of iterations (i.e. 10 iterations), and with large data sets to obtain good statistics for optimising the network functions.

Shao, Lin and Fossorier have proposed two stopping criteria [Shao et al, 99] , the “sign-change-ratio” (SCR) and “hard-decision-aided” (HDA) criteria, which are based on cross entropy [Buckley & Wicker, 99] .

In the SCR criterion, the average number of sign changes of LLRs between two consecutive iterations is calculated for all decoded information symbols. After each iteration, the average is compared to threshold levels obtained from simulations. Iterations proceed until the average falls below the specified threshold.

The HDA criterion compares the decoded information codewords after consecutive iterations to detect convergence. The decoder continues to iterate until two consecutive iterations generate identical hard values for the decoded information frame.

The authors state that both criteria are less complex than the cross entropy calculations. In addition, the SCR is computationally more efficient than the HDA criterion, but it performs slightly worse. According to their results, in the waterfall region, for a maximum of 6 iterations, both SCR and HDA criteria can save up-to 3 iterations, with a small degradation in error performance.

Recently, Matache, Dolinar and Pollara have defined three different stopping rules for turbo codes, which are the hard-decision, soft-decision and CRC rules [Matache et al, 00] .

When the hard decision rules are used, identical hard decisions after successive iterations or half iterations are monitored for terminating the decoder iterations.

The soft decision rules are based on the comparison of the LLR-derived metrics of each bit to a preset threshold at the decoder. Metrics are the average values of the absolute LLR values. Iterations proceed until the bit metrics are greater than the threshold.

The CRC method detects the erroneous decoded sequences using an outer CRC applied to the hard-decoded bits. The stopping condition with this rule is satisfied when the syndrome of the CRC code becomes zero.

In their report, when the upper limit of iterations is fixed at 20, the researchers have mentioned that the average number of iterations to vary between 4 and 7 for most combinations of the stopping rules described above. As further research, they have proposed developing a more analytical method of determining the thresholds used in soft decision rules.

This chapter introduces an alternative iteration control mechanism, namely the crossover codeword feedback system [Tanriover & Honary, 02b] , which has lower design and operational complexity than the neural networks, and utilises a user-defined BER threshold instead of metric thresholds for terminating iterations. The new technique uses the sign change of the LLRs as in SCR criterion, but it is based on the *progress history of the sign changes*, which can increase the metric accuracy.

The crossover codeword feedback system predicts the number of bit errors after the completion of a set of iterations, and triggers the decoder to perform further iterations unless the achieved BER is below a user-defined threshold. It will be shown that the soft value sign change patterns during iterations, which are identified by the new crossover coding algorithm, can be used to define a degree of reliability for each decoded information symbol.

The content and organisation of the remaining part of this chapter are as follows.

Section 4.2 includes an example, which demonstrates the significance of soft information, and the uncertainty in hard decision is discussed. In Section 4.3, a new algorithm developed for identifying the sign change patterns during iterations is introduced and an intuitive approach to defining uncertainty by using crossover codewords are presented. The codeword-bit error correlation is covered in section 4.4, where simulation data is interpreted in order to define error detection metrics, and the experimental data is used for deriving polynomials that represent the error-metric relationship. Coupling of the crossover detector with the turbo decoder is covered in section 4.5 in detail. Sections 4.6 and 4.7 deal with the performance and complexity evaluation of the proposed system, respectively. The last section in this chapter is the discussion, which briefly outlines the findings and the results.

4.2. Dimensions of Soft Information

The two dimensions of the soft information are its hard value determined by the sign, and its confidence level, determined by the magnitude. Suppose that the decoder has generated two soft values L_{e1}=+177 and L_{e2}=-58, for two received information symbols U_1 and U_2. Knowing that the BPSK modulation is used, hard values of U_1 and U_2 would be 1 and 0, respectively. It can also be seen that this hard decision is more likely to be correct for U_1 than it is for U_2, as the magnitude of L_{e1} is greater than L_{e2}.

Now assume that L_{e1} and L_{e2}, are obtained after 5 iterations, and let us consider one possible pattern of soft information progress for U_1 and U_2 (Figure 4.1). The iteration progress shows that the confidence level for U_2 monotonically increases in the negative region, whereas U_1 exhibits an irregular soft information progress. Knowing that after 5 iterations the hard decision is correct for both symbols, soft value progress in Figure 4.1 uncovers new information on both symbols. Even though, U_1 is correctly decoded after 5 iterations, its hard value consistently changes during iterations. For example, if iterations

were stopped after 4, U_1 would be decoded incorrectly. Similarly, a hard decision after the third iteration would result in the correct decoding of U_1. However, as the confidence progress of U_2 is monotonic, the hard values do not change during iterations. Therefore, for U_2, a hard decision after any iteration will still be correct.

If we were only given Figure 4.1, and no other information on U_1 and U_2, and were to apply a hard decision, the hard values of both symbols would be the same as before. However, most readers would be unsure about the decision on U_1, since an uncertainty is introduced by revealing the soft value progress. The uncertainty in this case arises from our knowledge about the history of the confidence values. The consistent sign change, termed as *the crossovers*, in the soft value progress of U_1 makes us reconsider its hard value, although its confidence is greater than L_{e2}. In this case, we cannot rule out the possibility that the hard value of U_1 is likely to change after another iteration.

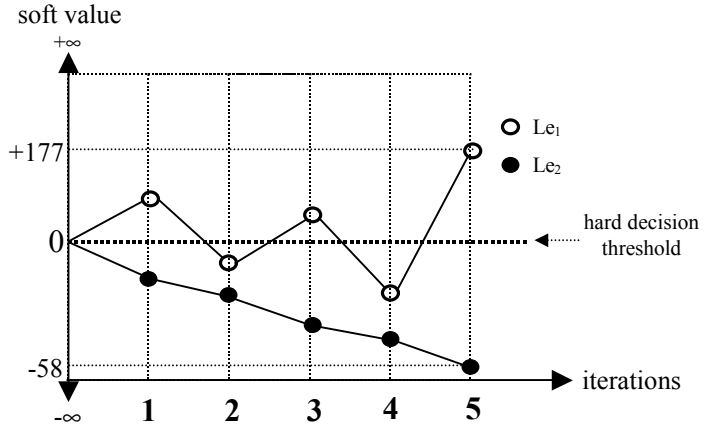


Figure 4.1 Soft value progress example

The example given above has covered only one of the many different soft value progress patterns that can be observed in the course of iterations in a turbo decoder. However, this example brings up a couple of important points to think about regarding the implications behind crossovers.

First, the crossovers can be related to decoder errors due to the uncertainty they introduce. In other words, the soft value progress pattern for an information symbol can be used as a measure of hard decision reliability of the same symbol [Tanriover & Honary, 99b] .

Secondly, as low confidence symbols are close to the decision threshold, crossover probability for those symbols is expected to be high. Ideally, a one-to-one relationship between the hard errors and the crossover locations is desirable for locating decoder errors accurately. However, such an expectation would be far fetched, knowing the random

characteristic of the channel noise profile and its impact on the modulated symbols and decoding [Tanriover & Honary, 00a] .

A more thorough analysis is necessary to obtain information on the crossover patterns in a turbo decoder as well as how they are associated with hard errors. Such an analysis urges unique identification of every possible crossover pattern for each decoded information symbol.

In the next section, a simple and effective method of representation, namely the crossover coding algorithm [Tanriover, 00d] , will be introduced to provide the right tool for our investigation.

4.3. Crossover Coding Algorithm

As explained earlier, a crossover is a case of sign change of the soft value of a decoded information symbol after consecutive iterations. Therefore, the crossover information can be expressed in binary format, since only the presence of sign change needs to be known. Table 4.1 summarizes the information provided by the crossovers. Note that, 'sign(L_{ei})' represents the sign of the soft information after the i^{th} iteration for an information symbol. If we wish to transform the binary information in the last column of the table into bits, '*yes*' and '*no*' can be replaced with 1 and 0, respectively, and the crossover state table can be structured as a trellis.

Each state in the two-state crossover trellis would represent the sign of the soft extrinsic value generated after an iteration for a symbol. Each branch on the trellis would correspond to a soft value sign transition between consecutive iterations, and each branch would produce one bit per transition (Figure 4.2). This trellis method would generate (I-1)-bit *crossover code words* after I iterations, and would allow us to numerically analyse the sign progress.

sign(L_{ei})	sign($L_{e(i+1)}$)	crossover
+	+	no
-	-	no
+	-	yes
-	+	yes

Table 4.1 Crossover states

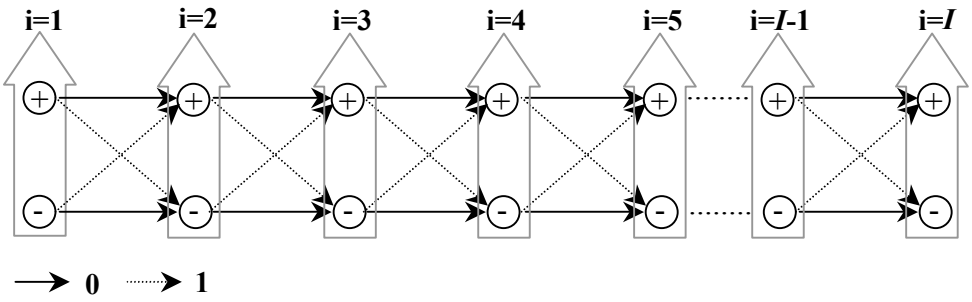


Figure 4.2 Crossover trellis

The crossover codeword generation will be illustrated with an example to clarify the method described above. Assume that after 8 iterations, for an information symbol, the set of soft values generated by the turbo decoder is given as {-23.45,+0.45,+53.01,-12.54,-6.89,-21.78,+15.93,+1.78}. We would like to find the corresponding crossover codeword for this progress pattern using a crossover trellis (Figure 4.3). Because the first soft output is negative, the starting trellis state will be negative. The second value, however, is positive and the resultant sign transition is represented with 1. After the third iteration, the soft value maintains its positive sign, and therefore this transition is indicated with 0. Bit assignment is carried out in this way until the 8th iteration, and the branch values of the trellis path gives the crossover codeword, which is 82 in decimal form.

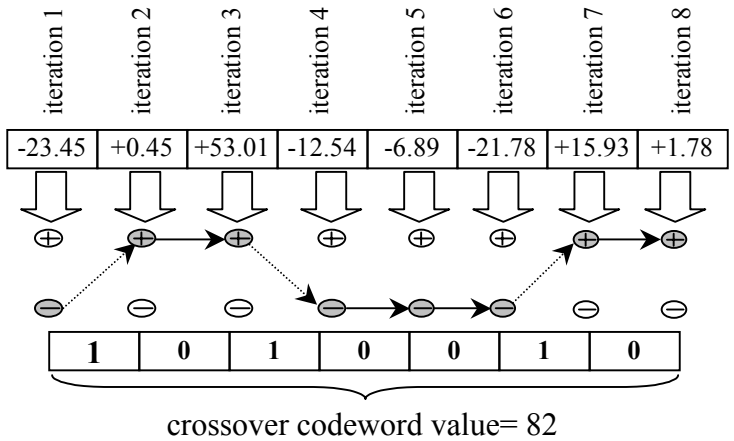


Figure 4.3 Crossover codeword generation example

Despite its simplicity, the crossover codeword algorithm is an effective tool for representing the sign change patterns in a simple format. As it was exemplified previously, changes in the soft value sign during iterations could be used to measure the level of uncertainty regarding the decoder hard decision. In order to explain the uncertainty-codeword relationship, let us consider a sample of codewords generated after *I* iterations (Figure 4.4).

Recall that uncertainty is related to the frequency of sign changes during iterations. In crossover coding terms, determining the sign change frequency is equivalent to calculating the Hamming weight of the generated codewords. In other words, large weight words would have a high degree of uncertainty as shown in Figure 4.4. When the transmitted symbols are not severely corrupted by the channel noise, iterations monotonically increase and stabilise the soft value magnitude of each information bit. As the channel noise increases, soft values fluctuate about the hard decision threshold during initial iterations and soft value sign changes still occur during later iterations. Therefore, *the zero code word has the least uncertainty whereas the codeword with I constituent ones has the highest degree of uncertainty*.

Codewords listed in Figure 4.4 represent a sub-space of the crossover codeword space. In reality, for I iterations, $2^{(I-1)}$ different sign change patterns are observed at the turbo decoder. However, the example chosen is a straightforward one in understanding the uncertainty-codeword association.

Note that the purpose of this analysis is to show the existence of a relationship between decoder hard errors and the soft value sign change patterns. As it was pointed out earlier, a one-to-one relationship between the crossovers and the errors does not exist. However, a high proportion of certain crossover codewords in a decoded information frame may be linked to the number of bit errors in that frame. The problem here is to determine which codewords would be the strongest indicators of the number of bit errors in a frame.

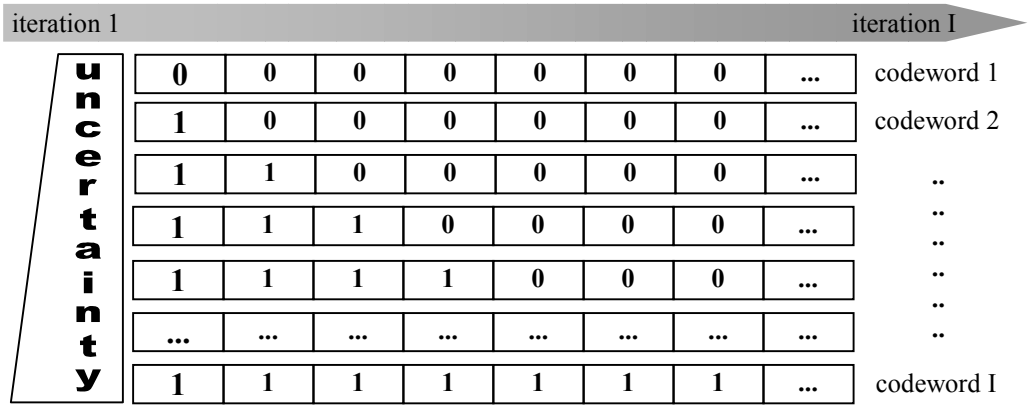


Figure 4.4 Crossover codewords and uncertainty

Even though the optimum solution of the above problem is not known, a valid sub-optimal approach can be deployed instead. Considering the uncertainty approach introduced earlier, a specific group of codewords can be selected from the codeword space.
The uncertainty argument can be refined with the following two important statements:

- *Localised distribution of ones towards the most significant bit (MSB) of any word can occur, as during initial iterations frequent sign changes are expected, and may not necessarily indicate an uncertainty.*
- *Long trails of zeros in a codeword, preferably close to the least significant bit (LSB), indicate improving soft value stability and, hence, decreasing uncertainty.*

The above two points, suggest that a crossover codeword needs to be considered in two parts; one part with Hamming weight that is greater than or equal to zero, and another part with zero weight. In order to put equal emphasis on both parts, a codeword is analysed in two equal sections.

Including all the constraints mentioned so far, the crossover codewords, generated after *I* iterations with high certainty, can be defined as *those words whose (I-1)/2 LSB have zero Hamming weight and the remaining most significant half has variable weight*. Any other codeword is considered to have a high level of uncertainty.

Figure 4.5 lists the possible codewords, generated after odd number of iterations between 3 and 15, which have low uncertainty levels. The reason for using odd number of iterations is that (*I*-1) needs to be divisible by two so that each codeword can be analysed in two equal parts. Note that the upper half of each codeword is allowed to be either 1 or 0 (indicated with an X), whereas the lower half consists of all zeros, as required.

		Codeword bit indices (13=MSB, 0=LSB)													
		13	***12***	***11***	***10***	***9***	***8***	***7***	***6***	***5***	***4***	***3***	***2***	***1***	***0***
iterations	**3**													X	0
	5											X	X	0	0
	7									X	X	X	0	0	0
	9							X	X	X	X	0	0	0	0
	11					X	X	X	X	X	0	0	0	0	0
	13			X	X	X	X	X	X	0	0	0	0	0	0
	15	X	X	X	X	X	X	X	0	0	0	0	0	0	0

Figure 4.5 Low uncertainty codeword groups for various iterations

4.4. Codeword-Error Correlation Analysis

So far a selection algorithm that filters codewords of low uncertainty is defined. In order to check the validity of this algorithm, a number of tests was performed. This section describes the test methods used and the metric definition.

4.4.1. Codeword and Error Data

When information symbol U_k, goes through I iterations, the crossover coding algorithm generates an (I-1) bit long crossover codeword, CW_k, which reflects the soft value sign change of U_k after consecutive iterations i_{I-1} and i_I (Figure 4.6). For CW_k to be a high certainty codeword ($CW_{k,H}$), it needs to satisfy the weight conditions explained in section 4.3.

The intended use of the high certainty crossover codewords (CW_H) is the bit error estimation of a turbo decoded information frame. By using a turbo decoder, the required error performance information was gathered according to the flowchart presented in Figure 4.7. The odd number of iterations between 3 and 15 were used as those provide sufficient convergence for the turbo code used. The E_b/N_o ratio was kept between 1.0 and 2.0 dB, which is within the waterfall region of the BER performance of the same turbo code. A total of 5000 information frames each with 4608 information symbols were decoded. The number of $CW_{k,H}$ per frame (n_{cwh}) and the number of bit errors (n_e) were counted and stored after decoding each frame.

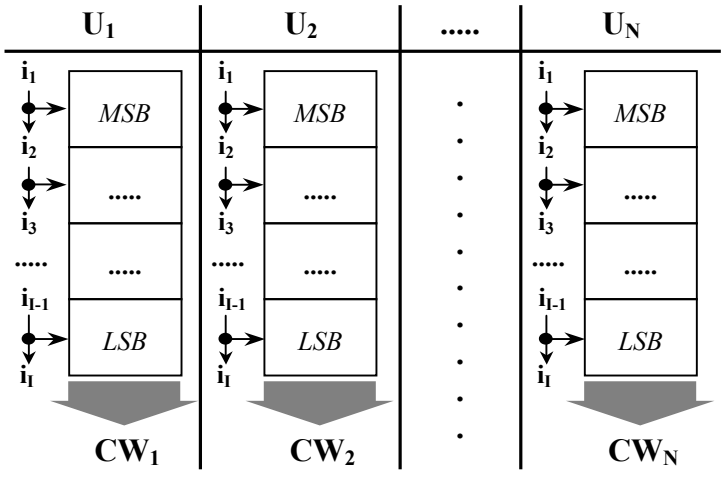


Figure 4.6 Codeword generation for information symbols after I iterations

Using the stored data, n_e was plotted against n_{cwh} for iterations applied. Results for 3, 5, 9 and 15 iterations are displayed in Figure 4.8a through Figure 4.8d.

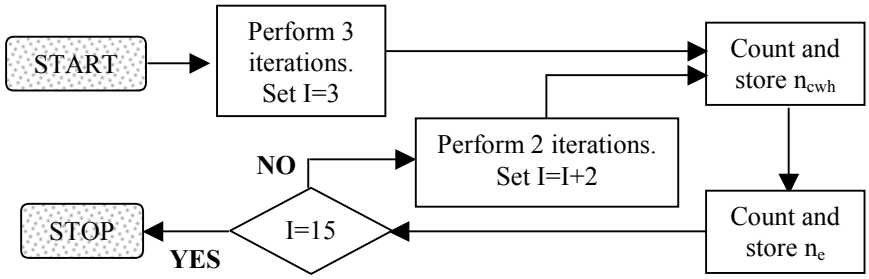


Figure 4.7 Codeword and bit error data collection at the turbo decoder

The first insight gained from the plots in Figure 4.8 is the existence of an inverse relationship between n_e and n_{cwh}. This observation justifies the selection criteria of the crossover codewords, as explained in the previous section. Since the CW_H were expected to be indicators of reliable decoding, they are inversely proportional to the number of bit errors in the decoded frames, i.e. high n_{cwh} means low n_e.

The second insight gained from the same data is that the codeword-error relationship changes with the number of decoder iterations. As the number of iterations increases, the inverse relationship becomes flatter and therefore, the spread of n_{cwh} increases. In order to clarify the final statement, consider Figure 4.8a and Figure 4.8d. For 3 iterations, bit errors range between 50 and 500 correspond to codeword totals within the 3000 and 4000 range. However, in the case of 15 iterations the same error range is related to the codeword totals that lie within 1000 and 4600. The reason for the variability of the n_{cwh} is due to discarding certain codewords as a part of the crossover codeword algorithm. This will be discussed in the next section.

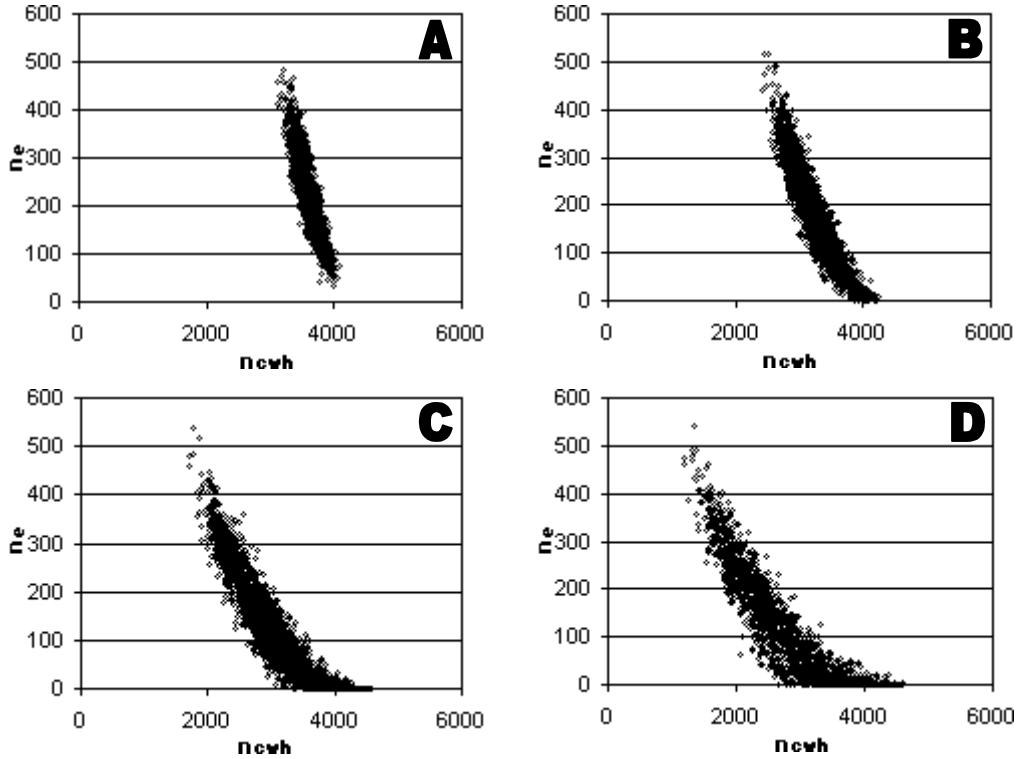


Figure 4.8 Error-codeword relation for **A-** 3 **B-** 5 **C-** 9 **D-** 15 iterations, E_b/N_o= 1.0 to 2.0 dB

4.4.2. Codeword-Error Relationship

When *I* iterations are applied at the turbo decoder, codewords of (*I*-1) bits are generated, which means that the total number of possible crossover codewords for an information symbol (T_{cw}) can be expressed as in (4.1).

Recall that for the high certainty codewords, the most significant (*I*-1)/2 bits can be either 1 or 0. Therefore, the total number of high certainty crossover words (T_{cwh}) is given by (4.2).

$$T_{cw} = 2^{(I-1)} \qquad \textbf{(4.1)}$$

$$T_{cwh} = 2^{(I-1)/2} \qquad \textbf{(4.2)}$$

The n_{cwh} values in all plots presented in Figure 4.8, are the total number of information symbols whose crossover codeword is *one of* the total T_{cwh} codewords. Therefore, in each plot T_{cwh}-T_{cw} codewords are ruled out as defined by the crossover algorithm. The ratio of T_{cwh} to T_{cw} is defined as the high certainty word ratio, r_{cwh}. Table 4.2 summarizes the *I*- r_{cwh} relationship. Notice how the r_{cwh} percentage decreases, as the number of iterations increases.

I	r_{cwh} (%)
3	50
5	25
7	12.5
9	6.25
11	3.13
13	1.56
15	0.78

Table 4.2 r_{cwh} percentages for odd number of iterations

The 1-r_{cwh} difference gives us the discarded codeword ratio, which increases with I. In other words, as I increases, the CW_H space becomes smaller. In this case, some of the codewords that are outside the codeword space can also indicate high certainty. The effect of discarding some of the high certainty codewords on the n_{cwh}-n_e relationship is going to be explained with an example.

Figure 4.9 is a simplified model of n_{cwh}-n_e relationship for three different high certainty codeword selections. The ranking of plots in ascending order of their r_{cwh} values is A, B and C. Put differently, 'A' and 'C' would represent the data for 15 and 3 iterations, respectively.

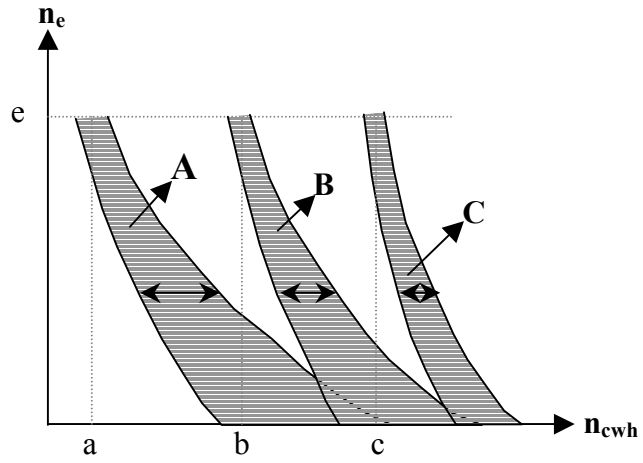


Figure 4.9 n_{cwh}-n_e relationship for three CW_H groups

For a constant n_e value, denoted e, the three plots have different n_{cwh} values, namely a, b and c. Notice that as r_{cwh} increases, the n_{cwh} value for n_e also increases. This shows that although the number of high certainty codewords decreases with the increasing number of errors, they may not correspond to correctly decoded symbols at all times. Therefore, when

more crossover patterns contribute to r_{cwh}, the probability of those patterns corresponding to errors also increases. This is clearly illustrated in Figure 4.9.

As the number of errors approach zero, the plots get wider, which means that the n_{cwh} spread increases. The reason for such a distortion in shape is again related to the size of the CW_H space and hence to r_{cwh}. In the case of plot 'A', the n_{cwh} that corresponds to n_e=0, has the largest spread since the CW_H space is the smallest, and a large number of high certainty codewords are not included in n_{cwh}. However, for plot 'C', higher number of crossover patterns are taken into account, which causes n_{cwh} to vary less as the number of errors decreases.

In its general form, the desired n_{cwh}-n_e relationship would be the one expressed in (4.3), where N is the frame size. Note that (4.3) would require such a CW_H space that all the constituent crossover patterns in it would correspond to correctly decoded information symbols, and the codewords outside the CW_H space would indicate the symbols in error, at all times. If such a codeword space had existed, then it would have been possible to correct all the decoding errors.

$$n_{cwh} = N - n_e \qquad \textbf{(4.3)}$$

The inverse relationship between n_{cwh} and n_e explained in this section is also valid under varying channel conditions. The n_{cwh}-n_e curve constructed for I iterations is also consistent with the metric data obtained for different noise levels.

In fact, data presented in Figure 4.8, reflects E_b/N_o changes between 1.0 and 2.0 dB. Decreasing channel noise, improves the turbo decoder's bit error performance (i.e. n_e decreases), which in turn increases the n_{cwh} metric value. In other words, varying channel conditions change the location of the (n_{cwh},n_e) ordered pair on the Cartesian coordinate plane according to the inverse relationship discussed earlier.

In the next section, mathematical representation of the n_{cwh}-n_e relationship will be explained, which will allow us to estimate n_e from n_{cwh}.

4.4.3. Polynomial Representation

Until now, it has been shown that n_{cwh} could be used as a metric to indicate the number of errors within a decoded frame, i.e. n_e. The inverse relationship between n_{cwh} and n_e, needs to be expressed as a function in the form of $f(n_{cwh}) = n_e$, before the crossover algorithm can be used in a turbo decoder.

Determining $f(.)$ is nothing more than finding the polynomial that fits best to the empirical data in the least squares sense. Curve approximations between the first and the fifth degree were investigated to find an appropriate mathematical representation of the codeword data.

Figure 4.10 shows the curve approximations for 3 and 15 iterations, which represent the two ends of the metric spread spectrum.

For 3 iterations (Figure 4.10a), all the polynomial approximations up to the fifth degree provide similar accuracy in representing the n_{cwh}-n_e relationship.

In the case of 15 iterations (Figure 4.10b), the linear approximation does not fit the data as accurately as the higher degree polynomials, and therefore can be disregarded. Except the first-degree function, mathematically, the remaining polynomials represent the data with near identical accuracy.

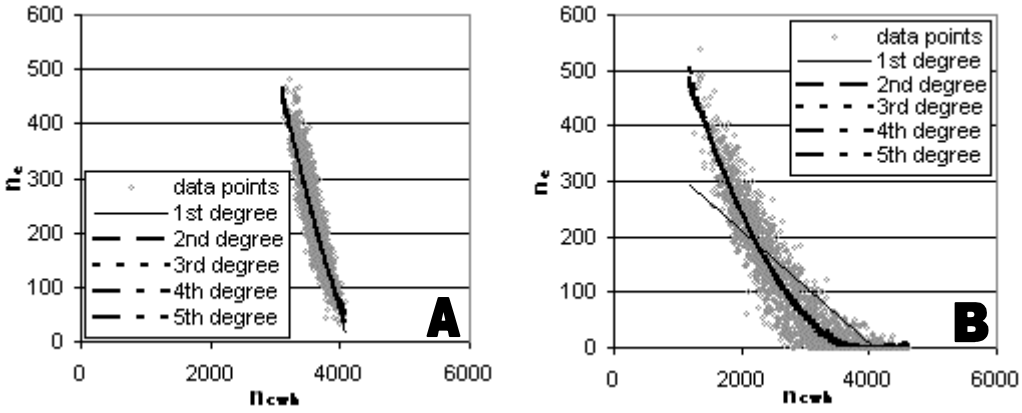


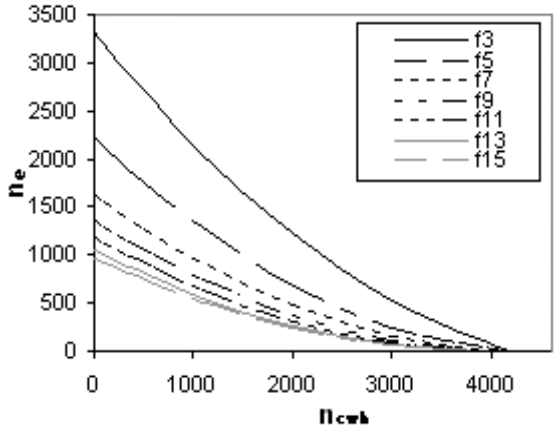
Figure 4.10 $f(n_{cwh})=n_e$ polynomial approximations *A*- 3 *B*- 15 iterations

Considering the accuracy and the computational complexity of the polynomials, first and second-degree approximations are preferred for 3 and 15 iterations, respectively.

For all the other odd numbered iterations, i.e. 5,7,9,11, and 13, the same polynomial search was repeated. It was observed that only for the 3-iteration case, could a linear function be

used, and for all the other iterations, second-degree polynomials provided sufficient accuracy.

For consistency in our analysis, all data gathered for different iterations, including 3 iterations, were represented by second-degree functions, $f_I(n_{cwh})$, where I denotes the number of iterations, to which f corresponds (Figure 4.11). Notice how the curves tend to overlap with each other as the number of iterations increases. This indicates that for iterations greater than 15, it is not necessary to gather experimental data for constructing approximation curves, which is a time-inefficient process. Instead, the second-degree function built for 15 iterations can be used for higher iterations, as the n_{cwh}-n_e relationship for those will be very similar to that obtained for 15 iterations. The closeness of the curves for 13 and 15 iterations verifies the last statement.



Second-degree polynomials

$f_3(n_{cwh})$=1.17e-4$(n_{cwh})^2$–1.28(n_{cwh})+3318.7
$f_5(n_{cwh})$=1.08e-4$(n_{cwh})^2$–0.98(n_{cwh})+2222.2
$f_7(n_{cwh})$=8.57e-5$(n_{cwh})^2$–0.75(n_{cwh})+1632.2
$f_9(n_{cwh})$=7.57e-5$(n_{cwh})^2$–0.64(n_{cwh})+1352.4
$f_{11}(n_{cwh})$=6.89e-5$(n_{cwh})^2$–0.57(n_{cwh})+1179.9
$f_{13}(n_{cwh})$=6.25e-5$(n_{cwh})^2$–0.51(n_{cwh})+1050.9
$f_{15}(n_{cwh})$=5.83e-5$(n_{cwh})^2$–0.48(n_{cwh})+960.54

Figure 4.11 Approximation curves and functions representing $f_I(n_{cwh})=n_e$

4.5. Turbo Decoder with Crossover Detector

The crossover codeword metric (n_{cwh}) has been shown to predict the number of bit errors within a decoded information frame. In this section, a novel iteration control mechanism, which uses the above detection metric, will be discussed as a part of a crossover detector block. The operation of the turbo decoder with the crossover detector will also be explained.

The coupling of the crossover detector block and the turbo decoder is presented in Figure 4.12. Assume that, a set of received information symbols, $\{U\}$, are turbo decoded, and the extrinsic soft information on set $\{U\}$, namely $L_e\{U\}$, is stored by the crossover detector during iterations.

The crossover detector block starts processing the stored soft information after three iterations and generates an iteration stop flag, F_I. If F_I=0, decoder performs two more iterations. Otherwise, it stops iterating and outputs the set of hard values $\{U`\}$.

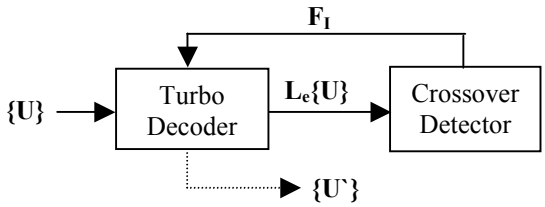


Figure 4.12 Turbo decoder with crossover detector

The crossover detector has four constituent blocks, which are FIFO (first-in first-out) buffer, crossover codeword generator, counter, and the error estimator and comparator (Figure 4.13).

The FIFO buffer stores the sign of the soft extrinsic information for symbol k, sign$[L_{e,k}\{U\}]$, that is processed by the detector block after every odd number of iterations except the first one.

The crossover codeword generator processes the sign change within the extrinsic value sign set, $\{\text{sign}[L_{e,k}\{U\}]\}$, for each information symbol. As described in section 4.3, the sign change is coded to generate crossover codewords. The high certainty codewords are sent to the counter.

Each received CW_H increments the counter, which is reset to zero before the counting begins. The counter value for an information frame, $\{U\}$, gives the n_{cwh} metric.

Error estimator block first determines the n_e value by substituting the n_{cwh} into the second-degree function that matches the number of completed iterations so far. Then, n_e is compared to an error threshold, E_t, that satisfies a user-defined bit error rate. If n_e is less than or equal to E_t, F_I is set to 1, which stops the iterations and causes the turbo decoder to perform hard decision. Otherwise, F_I is set to zero and the turbo decoder performs two more iterations before the crossover detector can perform further operations.

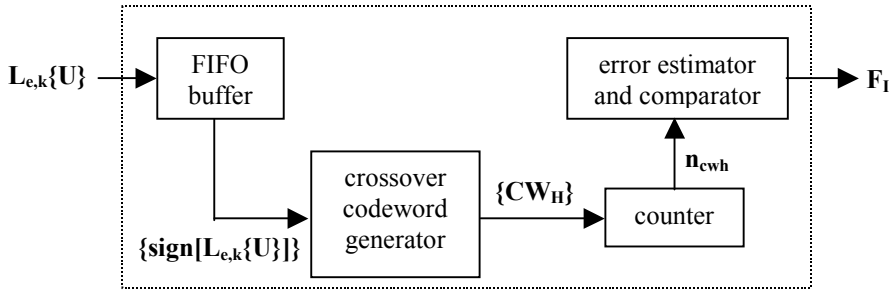


Figure 4.13 Crossover detector

The operation principle of the turbo decoder with the crossover detector has been outlined in Figure 4.14. In our system, the upper limit for the number of iterations is 15 to restrict the decoding latency. However, this limit can be increased without causing any problem to the proposed system operation (see section 4.4.3).

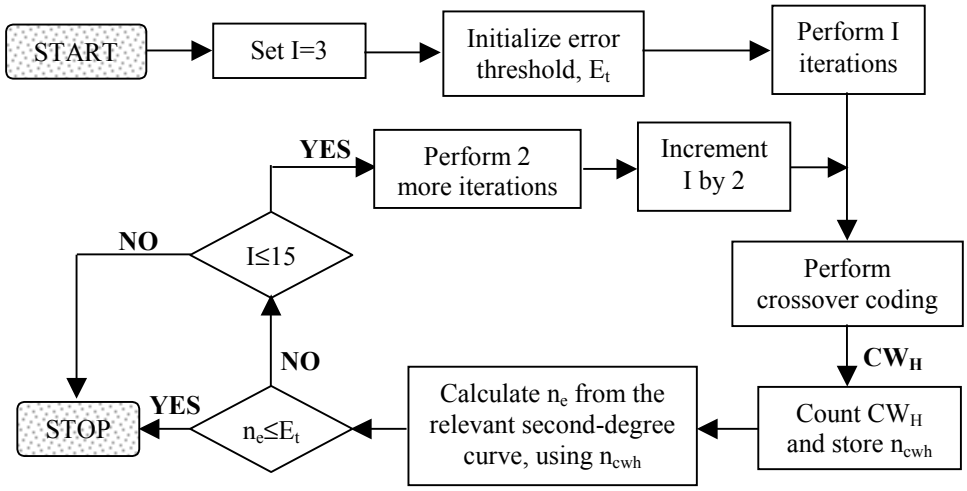


Figure 4.14 Decoder operation with crossover detector

The crossover detector is a supplementary tool for the turbo decoder, which attempts to keep the bit errors per frame below a user-defined E_t value. While doing this, it also increases the decoder processing efficiency by controlling the number of iterations per frame. In other words, it adjusts the number of iterations performed for each frame by monitoring the predicted BER.

4.6. Performance Evaluation

The crossover detector operates with a turbo decoder that processes frame lengths of 4608 bits. The decoding trellis has 4 states and decodes the g(7,5), half rate RSC code.

The performance of the proposed system, i.e. turbo decoder with the crossover detector, has been evaluated in terms of relative metric accuracy, error reduction and its iterative cost, and the bit error probability of the system. The results will be presented and discussed in this section.

The metric accuracy was tested for seven different E_t thresholds, which were preset in the detector. It should be noted that the metric accuracy is directly related to the approximation accuracy of the second-degree functions introduced in section 4.4.3. In other words, the difference between n_e and the actual number of bit errors within a frame (n_{fe}) indicates the detection metric accuracy. Figure 4.15 shows the relative detection error, Δ_{de} (4.4), within the 1-2 dB operation range for various error thresholds.

$$\Delta_{de} = \frac{\left| n_{fe} - n_e \right|}{n_{fe}} \qquad \textbf{(4.4)}$$

For each E_t, Δ_{de} varies in a particular pattern with changing noise level in the channel. As an example if we take E_t=50 curve, there are three operation regions for Δ_{de} variation that need to be considered, which are the negative slope (region A), the trough (region B), and the positive slope (region C) regions.

In region A, n_{fe} at the turbo decoder output is higher than E_t after initial iterations. Therefore, the crossover detector alarms the decoder to correct more errors by performing more iterations. As the high channel noise limits the reliability of the soft information within the turbo decoder, more iterations are required to reduce the number of decoder bit errors to E_t. When the number of iterations increases, the detector estimation frequency also goes up. Recall that each error estimation involves a second-degree approximation, which introduces errors. Frequent detector operations result in an accumulation of curve approximation errors that increases Δ_{de} under degrading channel conditions in the region A. It should be noted that as E_t decreases, i.e. lower BER is requested, the decoder is again triggered more frequently to perform further iterations, while operating in region A. This in turn increases Δ_{de} as seen in Figure 4.15.

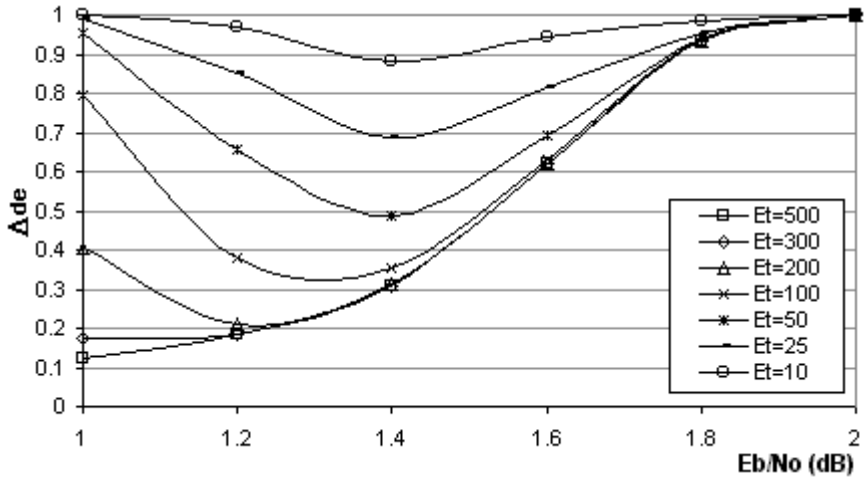


Figure 4.15 Relative detection error under varying channel conditions

Region B is the trough where the metric accuracy is the highest, i.e. Δ_{de} is at its minimum. In this region, the turbo decoder can reduce n_{fe} to the specified E_t with less number of iterations than in region A, due to the improved channel conditions. Notice that while E_t increases, the region B shifts towards left, as the E_b/N_o gets poorer. This is because the n_{fe} is close/equal to the E_t even at earlier iterations. Therefore, the detector is triggered less frequently or is not triggered at all, which reduces the curve approximation errors discussed above.

As we move from left to right in the region C, on one hand the turbo decoder bit error rate decreases, and on the other hand the number of decoder iterations approach to the minimum (i.e. 3), thanks to the improved signal level. In light of the previous explanations, increase in Δ_{de} in region C is not related to the curve approximations. In this region, when the detector estimates n_e, even a small absolute error (i.e.$|n_{fe}-n_e|$) in this estimation is going to result in a significant relative error since n_{fe} is also small.

From the discussion presented so far, the channel dependence of Δ_{de} can be summarized with the following remark:
Increasing relative error level with increasing channel noise in region A is a result of the cumulative effect of the curve approximation errors. However, relative error increase in region C with improving channel conditions is because the absolute error (although small) approaches n_{fe}. Region B is the point where error estimation is at the highest accuracy, due to the reduced adverse effects of factors introduced in regions A and C.

The system's error reduction accuracy depends whether for each information frame, n_{fe} is less than or equal to E_t once a hard decision is made for that frame. It should be pointed out that the error profile data presented in this section are the averages that are calculated after 60000 frame transmissions for each E_t. Figure 4.16 shows how the average frame errors are reduced below a user-defined error threshold over a range of channel conditions. When the E_b/N_o falls below 1.2 dB, for decreasing E_t the system fails to meet the specified error reduction. This is clearly observed for E_t<200 at 1 dB. However, the system manages to keep n_{fe} less than E_t within the 1.2-2.0 dB range. In addition to that, most of the time n_{fe} is kept much lower than the requested E_t upper limit. In fact, between 1.2 and 2.0 dB, the system provides between 33% to 88% higher error reduction than the requested E_t.

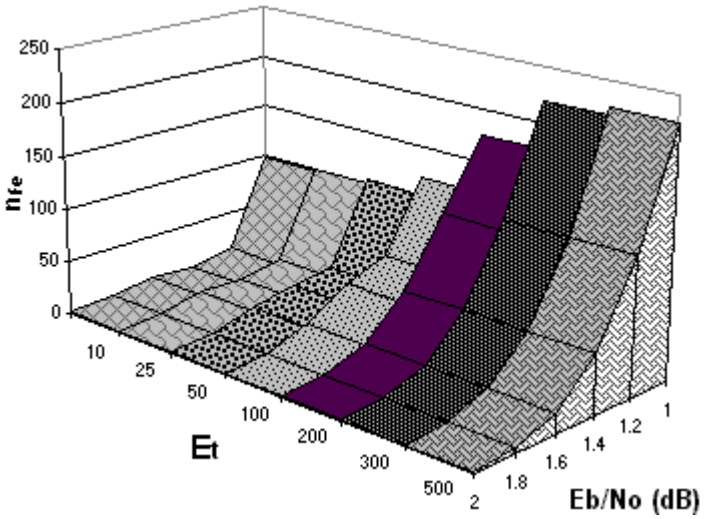


Figure 4.16 Error reduction accuracy

Figure 4.17 needs to be considered together with Figure 4.16, to get a view of the computational cost of the error reduction. As the channel noise increases, the average number of iterations per frame increases with decreasing E_t. This is because the decoder is triggered to perform more iterations to meet the $n_{fe} \leq E_t$ condition. In other words, as the soft information reliability per iteration declines when calculated from channel symbols that are excessively damaged by noise, the decoder needs to work harder to improve that reliability. The best example to this is when E_t=10 at 1.0 dB in Figure 4.17. The decoder performs about 7 iterations more than the minimum possible iterations, i.e. 3, to achieve the required error reduction, and yet it can only reduce n_{fe} to 100, which is 90 bit errors higher than E_t (Figure 4.16). Such inevitable failures are related to the code used in the system. Note that the bit errors can be reduced to E_t provided that E_t is set to a value that is within the error correction capability of the code, under a given channel condition. Similarly, if E_t is higher than what the code can actually achieve, the decoder does not have to perform too many iterations to meet the specified BER. Two distinct examples to such a case are when E_t is set to 300 and 500 as observed in Figure 4.17. Notice how the decoder performs only 3 iterations to keep n_{fe} less than E_t (Figure 4.16) within the 1.0-2.0 dB range. This observation will be addressed again, while discussing the BER performance.

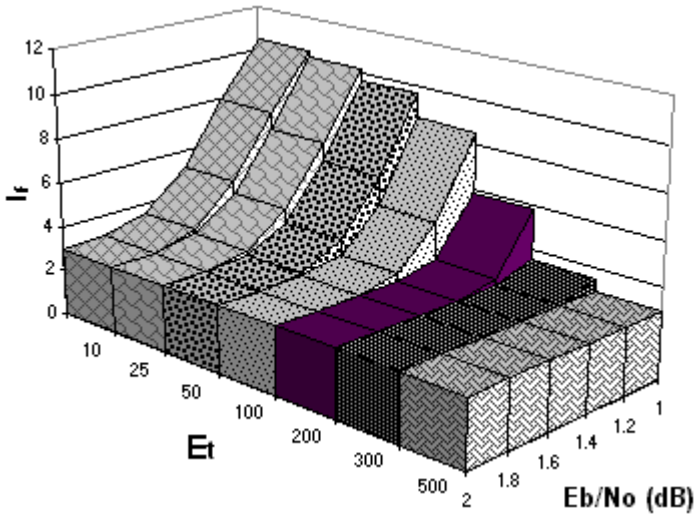


Figure 4.17 Average decoder iterations per frame

The objective of setting E_t to a constant value is to keep the BER below an upper limit (*UL*) over a range of channel conditions, by performing variable numbers of iterations that are controlled by the crossover detector (*CD*). Figure 4.18 shows how well this objective is achieved within the 1.0-2.0 dB region with various E_t thresholds. The BER upper limit for a given E_t is denoted *UL_*E_t. Decoder performance with a *CD* using a particular E_t is indicated with *CD_*E_t. On the same figure, the decoder performance for 3 and 15 iterations without a *CD* is also included (denoted *woCD*_3i and *woCD*_15i, respectively).

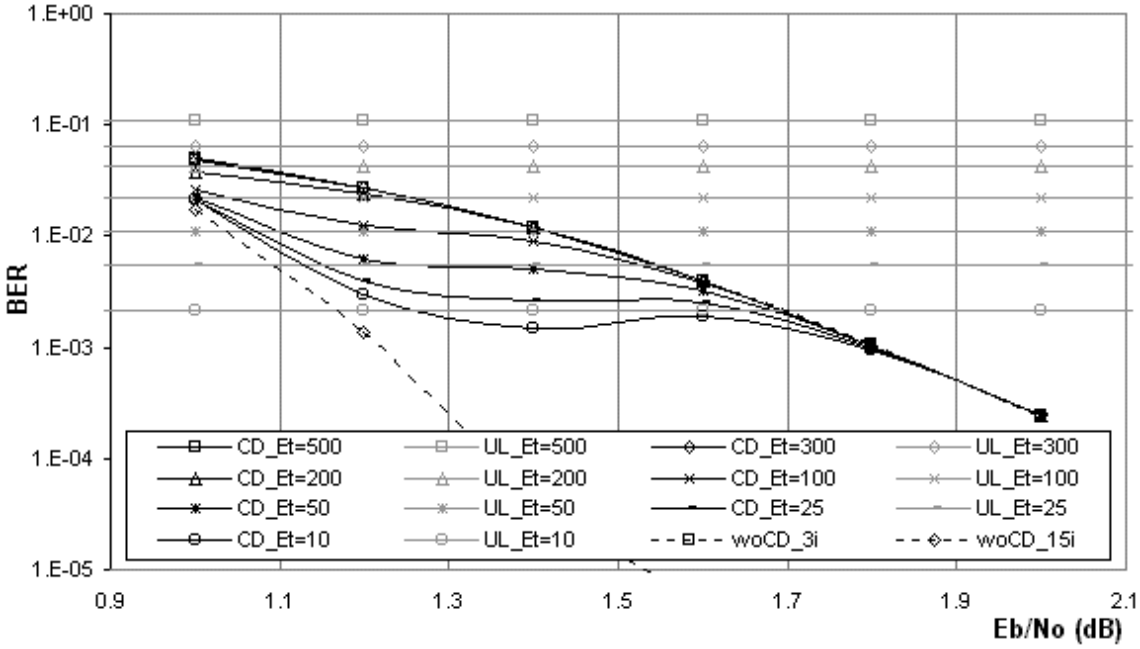


Figure 4.18 BER performance of turbo decoder with crossover detector

As expected, the BER performance of the system varies between *woCD*_3i and *woCD*_15i curves. It was discussed earlier that when E_t is higher than what the turbo code can achieve, the decoder needs to perform only 3 iterations to satisfy the user-defined bit error performance. This is also verified by the BER performance where the *CD_*E_t=300 and *CD_*E_t=500 curves overlap with the *woCD*_3i curve. Notice that the bit error performance

for E_t=300 and 500 is lower than the corresponding *UL* curves within the waterfall region. Decreasing E_t forces the turbo decoder to perform more iterations than 3 in order to lower the BER.

Each *CD_*E_t (particularly the ones for $E_t \geq 100$) curve in Figure 4.18 can be analysed in three distinct phases under improving channel conditions;

Phase 1: Convergence to 15 iterations

Phase 2: Stabilization

Phase 3: Convergence to 3 iterations

The bit error performance curves in Figure 4.18 will be interpreted according to a model presented in Figure 4.19 where all of the above phases are clearly marked for two E_t thresholds. Even though Figure 4.19 is an idealized representation of Figure 4.18, it is useful in explaining the trends in the BER plots of the turbo decoder with the crossover detector block in operation.

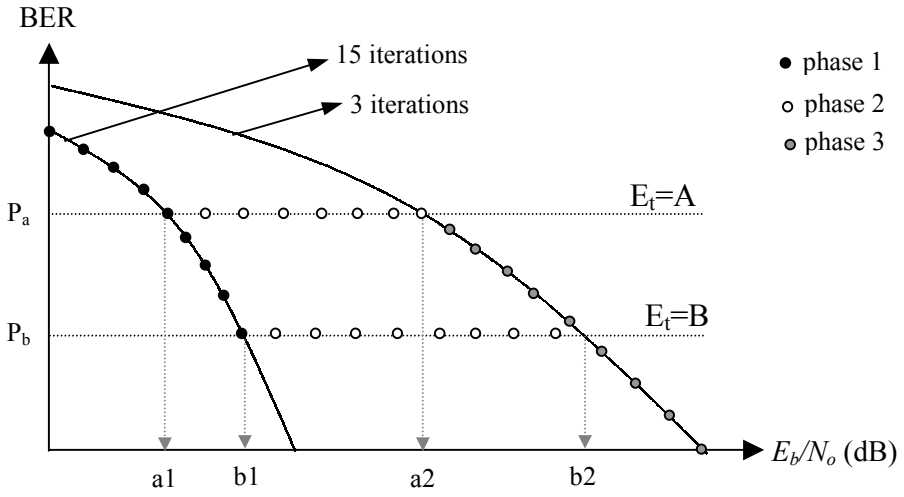


Figure 4.19 Average decoder iterations per frame

Phase 1 corresponds to the noisy channel conditions under which the decoder fails to achieve the BER specified by the E_t. The best thing that the decoder can do in this case is to perform the highest number of iterations allowed in the system (15 iterations in our system).

As the channel conditions improve, decoder slowly enters phase 2 where with less number of iterations the required BER is reached. Note that in phase 2, as the channel conditions improve, less number of iterations than that in phase 1 are needed to maintain a constant BER (see Figure 4.17).

In phase 3, the BER falls below the E_t requirement. In other words, the decoder reaches a lower probability of error than the required quality of service. While in phase 3, the decoder only needs to perform 3 iterations to achieve the specified BER.

4.7. Complexity Evaluation

The integration of the crossover detection block into the classical turbo decoder (Figure 4.12) introduces a slight increase in the overall processing complexity. In this section, the complexity overhead of the detector block is evaluated in comparison with the classical turbo decoder. Before a comparison can be made, the types of operations performed in each block needs to be identified. Note that the operations considered are the ones that have the largest share in decoding latency.

As it was discussed earlier, the crossover detector comprises four sub-blocks; FIFO buffer, crossover codeword generator, counter, and error estimator and comparator.

The FIFO block simply stores the sign of extrinsic information for each decoded information symbol. As the sign can be either positive or negative, it can be stored in binary format (e.g. 0 for negative and 1 for positive values). The size of the FIFO buffer needs to be NI, where N is the number of information symbols in a frame, and I is the number of iterations performed.

Crossover codeword generator (*CCG*) encodes the sign changes by using a 2-state trellis of depth I-1. The operations performed by this block are bitwise shifts while constructing the crossover codewords. For each information frame, $N(I\text{-}1)$ bit shifts are needed. In order to filter the CW_H from the crossover codewords, one logical AND operation per information symbol is required, which gives N logical AND operations per information frame. Note that the *CCG* block complexity depends on the number of crossover detector activations.

The counter block is simply incremented each time it receives one CW_H. The highest counter value can be N, which occurs when each information symbol generates a CW_H. If one increment can be regarded as one integer addition, then the maximum number of integer additions will be N per crossover detector activation.

The error estimator and comparator (*EEC*) block substitutes the final counter value, n_{cwh}, into the relevant second-degree functions (Figure 4.11) to calculate n_e, and to compare it to

the E_t. For I iterations, this block performs 6 floating point operations (4 multiplications and 2 additions) per information frame as well as one floating point comparison.

The total number of floating point operations carried out within the constituent component decoders comprises the largest share of the classical turbo decoder complexity. The number of floating-point operations performed depends on the number of trellis states (2^{K-1}) as well as the information frame size, N. Considering the state and branch probability calculations related to the max-log MAP algorithm, the total number of floating-point operations (including additions and multiplications) for a turbo decoder with two component decoders can be expressed as in (4.5).

$$2 \cdot I \cdot (16N + 2^{K+1}) \qquad \textbf{(4.5)}$$

In our system, the constraint length, K, is 3, which simplifies (4.5) to (4.6).

$$32 \cdot I \cdot (N + 1) \qquad \textbf{(4.6)}$$

Considering the information presented above, processing timings for the following types of operations can be used in our complexity evaluation: floating point comparison (*comp*), bitwise AND (*and*), bitwise shift (*shift*), integer addition (*add_int*), and floating point additions and multiplications (*fpo*). The timing for each operation has been measured using stopwatch functions available as libraries in standard C programming language that provide up to 1 microsecond accuracy.

In order to express the complexity analysis independent from the type of processors used, timing measurements are presented relative to the *and* operation, which introduces the least processing delay, T. Table 4.3 presents the processing delays in units of T.

In order to compare the crossover detector complexity with the turbo decoder's, information in Table 4.3 needs to be jointly considered with the operations performed in each block described above. As the operational complexity for each block is iteration and/or activation dependent, the upper and lower limits of computational load are calculated to provide comprehensive information. This is done by considering the maximum and minimum values for 'I' and the number of activations (A) (I_{min}=3, I_{max}=15 and A_{min}=1, A_{max}=7) for each block.

Operation	Delay (*T*)
and	*T*
shift	1.4*T*
comp	18.5*T*
add_int	*T*
fpo	7.1*T*

Table 4.3 Normalised operation delays for crossover detector

The *CCG* block spends $N(I\text{-}1)(1.4T)$ of processing time for *shifts* and *NTA* for *and* operations per *A* activations. In our system, *I* can take any odd value between 3 and 15, inclusive. Therefore, the maximum processing time spent on *shifts* is when *I* takes all odd values between 3 and 15. Similarly, the number of *and* operations will increase with the number of block activations, which can vary from 1 to 7. Table 4.4 summarizes the upper and lower complexity limits for the *CCG* block in terms of *N* and *T* (Note C_x denotes the complexity function for block *x*).

I and *A* values	Complexity Function
I=3,5,...,15, A=7	$\max\{C_{CCG}\}=(56N)(1.4T)+7NT=85.4NT$
I=3, A=1	$\min\{C_{CCG}\}=(2N)(1.4T)+NT=2.8NT$

Table 4.4 Maximum and minimum *CCG* processing time

The counter operation time is equal to *NTA* per *A* activations. Table 4.5 shows the processing time limits for the counter block.

A values	Complexity Function
A=7	$\max\{C_{counter}\}=7NT$
A=1	$\min\{C_{counter}\}=NT$

Table 4.5 Maximum and minimum counter processing time

For the *EEC* block the *fpo* and *comp* operations take $6(7.1T)A$ and $(18.5T)A$, per *A* activations, respectively. Table 4.6 lists the processing times for the *EEC* block.

The turbo decoder processing time is given by $32I(N+1)(7.1T)$ as its complexity is heavily due to the *fpo* operations as described earlier. The upper and lower processing time limits for the decoder are presented in Table 4.7.

A values	Complexity Function
A=7	$\max\{C_{EEC}\}=(42.6T)7+(18.5T)7=427.7T$
A=1	$\min\{C_{EEC}\}=42.6T+18.5T=61.1T$

Table 4.6 Maximum and minimum *EEC* processing time

It was mentioned in the beginning of this section that the complexity overhead of the detector block would be evaluated in comparison with the classical turbo decoder's. Having obtained processing time figures for the *CD* constituent blocks and the turbo decoder, Table 4.8 is constructed where all the complexity figures are combined and presented together.

I values	Complexity Function
I=15	$\max\{C_{decoder}\}=480(N+1)(7.1T)=3408T(N+1)$
I=3	$\min\{C_{decoder}\}=96(N+1)(7.1T)=681.6T(N+1)$

Table 4.7 Maximum and minimum decoder processing time

The final step of the analysis is to calculate the numerical C_{CD} to $C_{decoder}$ ratio, which is independent of the *N* and *T* variables. It can easily be seen that for such a ratio, variable *T* cancels out as it appears in both the numerator and the denominator terms. In order to discard term *N*, the limit condition where *N* approaches infinity will be considered.

The limit condition in (4.7) is shown for the maximum delay only. The same method can be applied in the minimum case as well. In the case of the minimum delay, the limit ratio is equal to 0.0056. In other words, the upper and lower processing time limits for the *CD* block varies between 2.71% and 0.56% of the turbo decoder's complexity, respectively.

	maximum delay	minimum delay
C_{CD}	$(92.4N+427.7)T$	$(3.8N+61.1)T$
$C_{decoder}$	$3408T(N+1)$	$681.1T(N+1)$

Table 4.8 *CD* and decoder processing time expressions

$$\lim_{N\to\infty}\frac{C_{CD}}{C_{decoder}}=\lim_{N\to\infty}\frac{92.4N+427.7}{3408N+3408}=0.0271 \qquad \textbf{(4.7)}$$

4.8. Discussion

In this chapter, an iteration control mechanism that utilises the soft output progress of the turbo decoder has been introduced.

It has been shown that the soft value sign change patterns during iterations, which are identified by the crossover coding algorithm, can be used to define a degree of certainty for each decoded information symbol. The number of high certainty codewords within a frame was found to be inversely proportional to the number of bit errors within that frame.

The crossover codeword feedback system uses this inverse relationship to predict the number of bit errors after the completion of a set of odd numbers of iterations, and triggers the decoder to perform more iterations unless the achieved BER is below a user-defined threshold.

The introduced system stabilizes the BER performance while efficiently allocating the processing power available to the turbo decoder. Instead of blindly performing a constant number of iterations, the turbo decoder can externally be controlled by a more intelligent block that allows it to stop when a specified BER condition is satisfied.

The decoding delay introduced by the crossover detector increases the inherent turbo decoding latency by an amount within the 0.56%, 2.71% range, depending on how frequently the detector is activated.

Chapter 5

Post-Decoding Channel Estimation

5. Post-Decoding Channel Estimation

5.1. Introduction

The objective of error correction is to reduce or eliminate the corrupting effects of the channel noise on the transmitted information. In order to do that, noisy information and the parity symbols are processed according to a decoding algorithm. In the case of turbo codes, when either the MAP algorithm or one of its sub-optimal versions, i.e. log-MAP or max-log-MAP, is used, a priori channel information is required. Each transmitted information frame is exposed to random and independent noise levels. Therefore, the channel state information needs to be estimated for each frame before decoding can commence.

So far, the channel estimation techniques developed for turbo codes are based on pilot symbol insertion (PSI), and/or decoder derived methods (DDM).

Valenti and Woerner [Valenti & Woerner, 98] , [Valenti & Woerner, 99] have suggested to transmit pilot symbols that are used for obtaining the initial channel state information prior to decoding. In their work, the channel information is *refined* after each decoder iteration, and hence the decoder error performance is improved. When BPSK modulation is used, the authors claim to achieve a BER performance as close as 1.5 dB to the ideal case (at BER=10^{-4}) for flat-fading channels. Their simulation results also show that the PSI method by itself is 2.5 dB away from the ideal case.

Coulton and Honary [Coulton & Honary, 98] have introduced a noise variance estimation scheme applicable to AWGN channels. They have introduced three different metrics, which are calculated from the minimum Euclidean distance between a received codeword and the possible transmitted codewords. With this estimation technique, a simple look-up table that relates a calculated metric value to the channel noise, is used. According to their results, the new technique performs 0.25 dB worse than the ideal (at BER=10^{-4}), and can yet be improved using an iterative method as in [Valenti & Woerner, 98] .

Some researchers have also compared the iterative and non-iterative channel estimation methods for flat-fading channels. Kim et al [Kim & Yoon, 00] obtained initial estimates of the fading amplitude by using a filter function and optimise this estimate after each iteration. Their iterative estimation technique performs 0.5 dB worse than the ideal and 0.2 dB better than the non-iterative technique (at BER=10^{-4}) for short frame sizes (i.e. 450 information symbols).

All of the estimation techniques mentioned above derive the required channel state information prior to decoding. However, considering the sensitivity of the turbo decoder to channel estimation errors, alternative techniques can be developed. Summers and Wilson [Summers & Wilson, 98] have studied the sensitivity of the decoder performance to misestimation of the SNR and have proposed a statistical method for determining the SNR. Their simulations over an AWGN channel show that when E_b/N_o is estimated with a ±0.5 dB error, the BER of the turbo decoder is not affected at all. More importantly, as the channel conditions improve, the sensitivity of the turbo decoder to misestimating decreases. The authors also mention similar independent results obtained by other researchers [Jordan & Michols, 96] ,[Reed & Asenstorfer, 97] .

Hoeher [Hoeher, 97] has also shown that perfect channel estimation for AWGN channels is not necessary for accurate turbo decoder operation. Based on simulation results, he further argues that by fixing the E_b/N_o in the middle of the waterfall range instead of estimating it prior to decoding, almost no degradation in the bit error performance is observed.

In this chapter, an alternative estimation technique for time-varying channels is introduced. This technique has been tested assuming AWGN, by varying the signal-to-noise ratio to investigate the tracking performance of the algorithm. Thus the applicability of this technique to fading channels has been explored. The new method will be shown to provide sufficient estimation accuracy to achieve near-ideal BER performance at the expense of a small additional decoding delay. As it will be explained, the channel information calculated after decoding one information frame will be used for decoding the following frame - a method that exploits the decoder's sensitivity level to estimation errors. The remaining sections of this chapter are organised as follows.

Section 5.2 describes the channel estimation problem in iterative decoding. Section 5.3 highlights the interdependence of the soft extrinsic information and the channel noise, which is followed by the introduction of a simple identification method that indicates the extrinsic information progress for each symbol in section 5.4. The channel estimation metric definition and the channel initialisation method are presented in sections 5.5 and 5.6, respectively. Incorporation of all the channel estimation components into an estimator block that works with the turbo decoder is covered in section 5.7. Sections 5.8 and 5.9 constitute the last two parts of channel estimation, where channel estimator performance and its computational complexity are evaluated.

5.2. Channel Estimation Problem in Turbo Codes

The MAP algorithm is based on the calculation of branch (γ) and state (α and β) probabilities on the decoding trellis structure. The α and β probabilities are calculated recursively from the γ probabilities, which are functions of the channel reliability factor, L_c. The channel reliability factor [Hagenauer et al, 96] is calculated from (5.1), where '*a*' denotes the fading amplitude and E_s and N_o are the symbol energy and the single-sided power spectral density of white noise, respectively. In the case of AWGN channel, '*a*' is set to unity, as there is no fading.

$$L_c = 2 \cdot a \cdot \frac{E_s}{N_o/2} \qquad \textbf{(5.1)}$$

In the detection process, the noise of interest is the one at the matched filter output [Sklar, 88] . Therefore, the variance of the correlator output gives us the noise variance, σ^2 (5.2) for the band-limited white noise.

$$\sigma^2 = \frac{N_o}{2} \qquad \textbf{(5.2)}$$

Conventionally, E_s is expressed in terms of energy per bit (E_b) by using $E_s=E_bR$, where R is the code rate. If E_s is normalised to one, E_b/N_o ratio can be expressed as in (5.3).

$$\frac{E_b}{N_o} = \frac{1}{2\sigma^2 R} \qquad \textbf{(5.3)}$$

Substituting the ratio in (5.3) into (5.1) gives us the relationship between the L_c and the σ^2 for the AWGN channels (5.4).

$$L_c = \frac{2}{\sigma^2} \qquad \textbf{(5.4)}$$

As unveiled by (5.4), estimating channel conditions for a transmitted frame is equivalent to determining the noise variance, σ^2, at the matched filter output.

5.3. Channel Dependency of Extrinsic Information

One of the reasons for turbo decoder's excellent bit error performance is its SISO structure. Extrinsic information is passed between the component decoders to progressively improve the confidence level of the information symbols in a frame.

The accuracy of the iterated extrinsic information directly influences the error performance of the turbo decoder. In addition, the reliability of the extrinsic information is related to the level of added noise to the received symbols. Therefore, it can be inferred from the previous statement that for an information symbol k, the progress of extrinsic information (L_{ek}) is expected to vary with changing noise levels in the transmission channel.

In order to visualize how L_{ek} progress relates to the channel conditions, the iterative extrinsic soft value progress patterns of a number of information blocks were observed for 8 iterations. The sample data are presented in Figure 5.1, where three different channel conditions are considered for a randomly selected information frame.

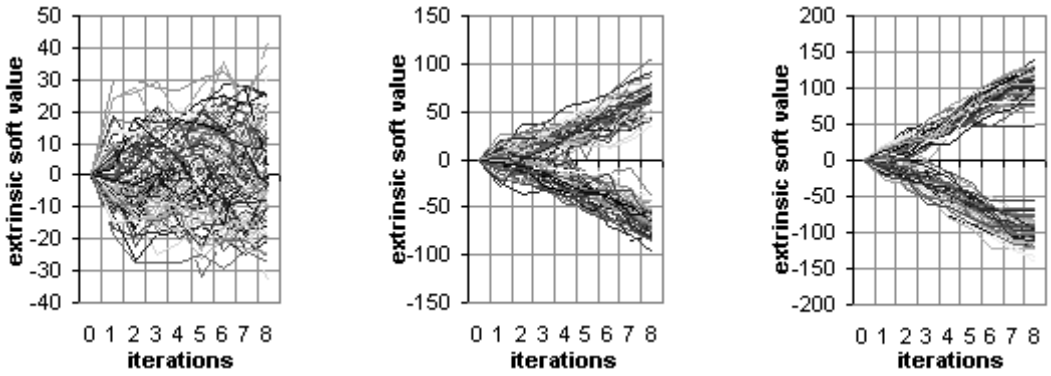


Figure 5.1 Extrinsic soft values for E_b/N_o= **A**- 0.5 dB **B**- 0.8 dB **C**- 1.1 dB

Clearly, the L_e progress patterns are differentiable for different noise levels, which highlight the presence of the L_e-noise correlation.

More importantly, one aspect of the progress patterns illustrated in Figure 5.1 is particularly interesting from a channel estimation point of view. As the noise level increases, the magnitude of L_e fails to follow a *monotonic increase* trend (Figure 5.1a). However, as the channel conditions improve, the magnitude progress tends towards being incremental (Figure 5.1c).

The above observation is significant in the sense that the ratio of information symbols with monotonically increasing L_e magnitude to the total number of symbols (i.e. the frame length), can provide information on the channel conditions. Data in Figure 5.1 also supports this argument. For 8 iterations, L_e magnitude can change 8 times (L_e values are initially set to zero). When we calculate the percentage of symbols with 8 successive magnitude increases for 0.5, 0.8 and 1.1dB, we obtain 0.1%, 2.1% and 23.7%, respectively. A similar trend is also observed for other randomly chosen frames.

The observations mentioned so far can be summarized with the following remark: "*The proportion of information symbols that have monotonic L_e magnitude incremental progress during iterations increases as the channel noise decreases*".

Even though L_e iterative progress is related to the channel conditions, further analysis is necessary to develop a L_e-derived channel estimation method. In the next section, a simple method used for identifying progress patterns will be explained, which will form the basis of our analysis.

5.4. Extrinsic Value Progress Pattern Identification

The magnitude changes of L_e values during iterations need to be analysed for each symbol before reaching a conclusion about their channel dependence.

For this purpose, the parallel turbo decoding structure was used as shown in Figure 5.2. It should be noted that in the decoder model, interleaver and de-interleaver blocks have been excluded to simplify the representation, although they are a part of the system. In the parallel model there are two pipelines that operate simultaneously; pipeline 1 that is initiated by component decoder 1, and pipeline 2 initiated by component decoder 2. In each pipeline, both component decoders operate once per iteration. Additionally, two new structurally identical blocks have been incorporated into the decoder structure; progress word generators (*PWG*) 1 and 2.

For each information symbol, the *PWG* block monitors the magnitude progress of the extrinsic information in a pipeline, and encodes this progress in binary format. The *PWG* block has three constituent blocks, which are a comparator, shift register and a word weight counter (Figure 5.3).

The comparator has two input terminals that accept soft information, which are the a priori information for symbol k coming from component decoder j, denoted $L_{uk,j}$, and the extrinsic information, $L_{ek,i}$. Notice that the L_u and the L_e values come from different component decoders, i and j, which are on the same pipeline (Figure 5.2). If the terminal 1 input *magnitude* of the comparator is greater than that of terminal 2, its output becomes 1, and otherwise, output is zero.

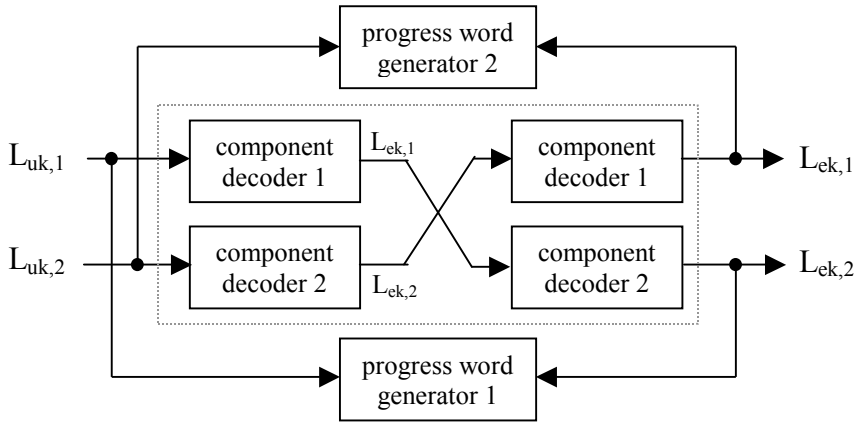


Figure 5.2 Extrinsic progress analysis model for parallel decoding

The comparator is followed by *N* shift registers, where *N* is the information frame size. Each shift register length is equal to the total number of iterations *(I*). Binary values generated by the comparator are stored and shifted right after each iteration. As one bit per iteration is generated in each *PWG*, at the end of *I* iterations an *I*-bit progress codeword (PW_k) is constructed for each information symbol, *k*.

After the completion of *I* iterations, each PW_k is passed to a counter, which calculates the Hamming weight of that codeword. The weight of each PW_k can be between 0 and *I*. The counter also stores the weight distribution of all the progress words.

The Hamming weight of a PW_k tells us how many times the L_{uk} magnitude is amplified during its journey through a decoder pipeline. This is measured by the difference between the L_{ek} and L_{uk} for each information symbol, as explained above.

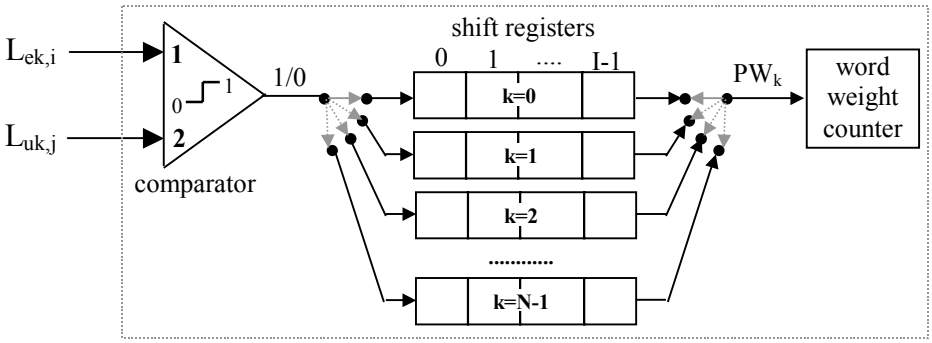


Figure 5.3 Progress word generator model

In other words, *the Hamming weight of a progress codeword is a measure of the acquired confidence level during iterations, such that high weight indicates a high confidence level for an information symbol*. Note that for each *k*, two *I*-bit codewords are generated in the turbo decoder - one word per *PWG* (Figure 5.2).

The next step in the analysis is to observe the extrinsic magnitude progress by observing the codeword distribution for various channel conditions and to uncover the relationship between the codeword weight and the channel noise. The information frame size, *N*, is set to 3072 for a 4-state, 1/3 rate code for the turbo code used in the model. The component decoders use max log-MAP, and the maximum number of iterations, *I*, is set to 8. While gathering the data, the turbo decoder is assumed to have perfect knowledge of the σ^2- a condition that is necessary to increase the accuracy of the extrinsic information calculations. For each simulated E_b/N_o ratio, 20000 information frames are decoded, and the codeword weights are averaged.

The weight distribution in Figure 5.4, presents the symbol percentages with all possible total weights between 0.4 and 1.2 dB. The total weight axis represents the sum of codeword weights from both *PWG* blocks. Weights higher than or equal to 8 are grouped under the '*8+*' category. It can clearly be seen that decreasing channel noise shifts the weight distribution of each symbol towards the *8+* region, and symbols with weight 7 or less start to disappear.

Findings presented until now indicate that information on the progress codeword weight distribution can be used to define a channel estimation metric, which is discussed in the next section.

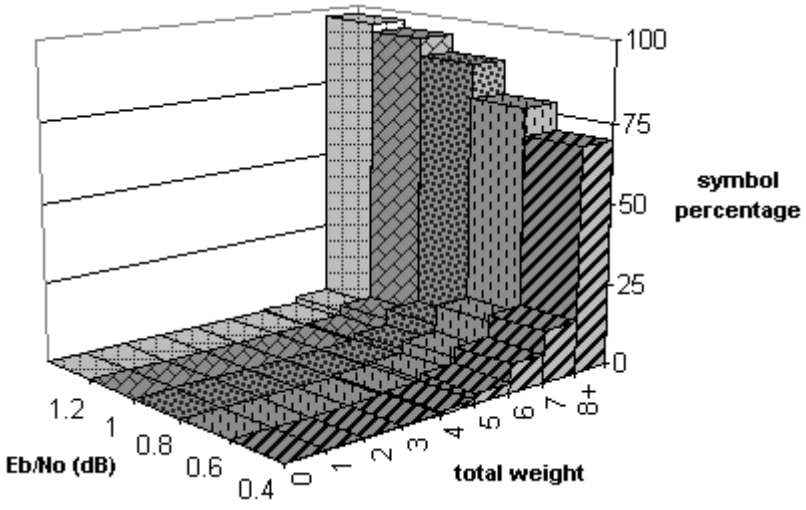


Figure 5.4 Progress word weight distribution

5.5. Channel Estimation Metric Definition

As described in the previous section, the extrinsic value progress pattern, quantified by the progress codeword weights, gives an important clue to the channel noise level. Especially the ratio of symbols with minimum weight 8 to the total number of symbols in a frame can

be a reliable indicator of noise level in the channel. It is therefore appropriate to define the channel estimation metric (m_{ce}) as in (5.5). Note that $N(w_{8+})$ denotes the number of symbols with weight 8 or greater, and N is the total number of symbols within an information frame.

$$m_{ce} = \frac{N(w_{8+})}{N} \qquad \textbf{(5.5)}$$

Having defined a channel metric, a few important points related to the channel estimation strategy needs to be emphasized. Let us first consider the metric variation under different noise levels observed for each frame.

Figure 5.5 presents a set of m_{ce} values for a sample of 50 frames transmitted at 0.4, 0.8 and 1.0 dB. It can be seen that the m_{ce} has a scattered characteristic especially at low signal-to-noise ratios. The σ^2 characteristic is also expected to have such a scattered characteristic, assuming that it is related to m_{ce}. Note that m_{ce} is obtained after decoding and therefore it cannot be used for decoding the information frame, from which it is obtained. One way of using this metric is to store it for decoding the next frame, i.e. with a delay of one frame. However, since the m_{ce} values may vary significantly from one frame to another, such an approach would result in high degrees of estimation errors. It is therefore necessary to introduce an effective method that will allow using m_{ce} with minimal estimation errors.

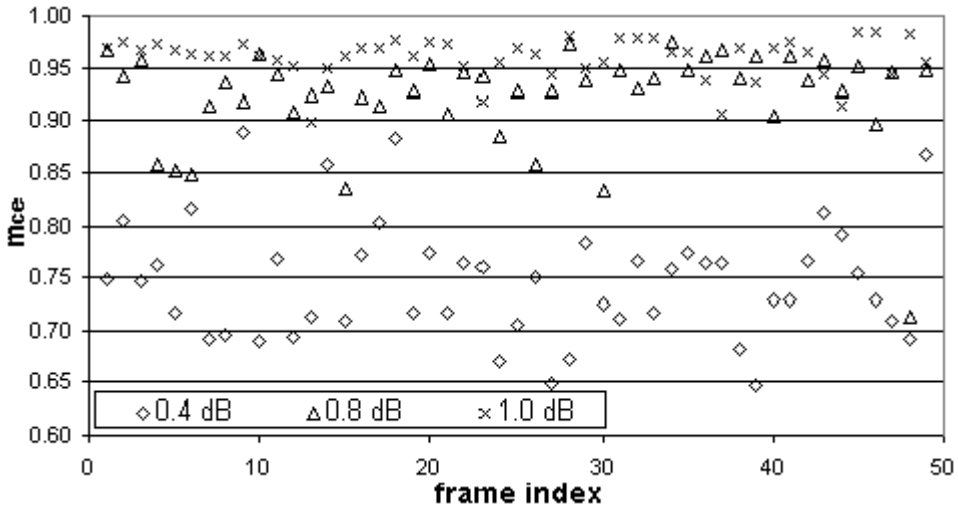


Figure 5.5 Metric stability

When Figure 5.5 is revisited, it can be seen that m_{ce} values for each E_b/N_o are distributed around different averages. In order to unveil this metric trend, filtering can be applied. Due to its effective smoothing characteristics and simple structure, a moving average filter is often used for analysing the trend in a set of scattered data, and is therefore preferred here.

The transfer function of a moving average filter of length l is given in (5.6), where $x(n)$ and $y(n)$ are the n^{th} input and output, respectively.

$$y(n) = \sum_{i=0}^{l-1} x(n-i) \qquad \textbf{(5.6)}$$

In order to demonstrate the m_{ce} stability, metric data are smoothed by a moving average filter with l=10 (Figure 5.6). Further smoothing can be achieved by simply increasing the filter length. However, increasing l may have disadvantages depending on the channel conditions, which will be explained later.

The important insight gained from filtered m_{ce} (m_{cef}) data is that for an average E_b/N_o value, the metric is distributed around a near-stable average value. As the channel conditions improve, evidently, the metric stability increases. Even though the m_{ce} is not a real-time channel estimation metric, when filtered, it can provide information on channel variations. This trend information is useful for decoding an information frame and can be updated by the extrinsic information of the same frame after decoding.

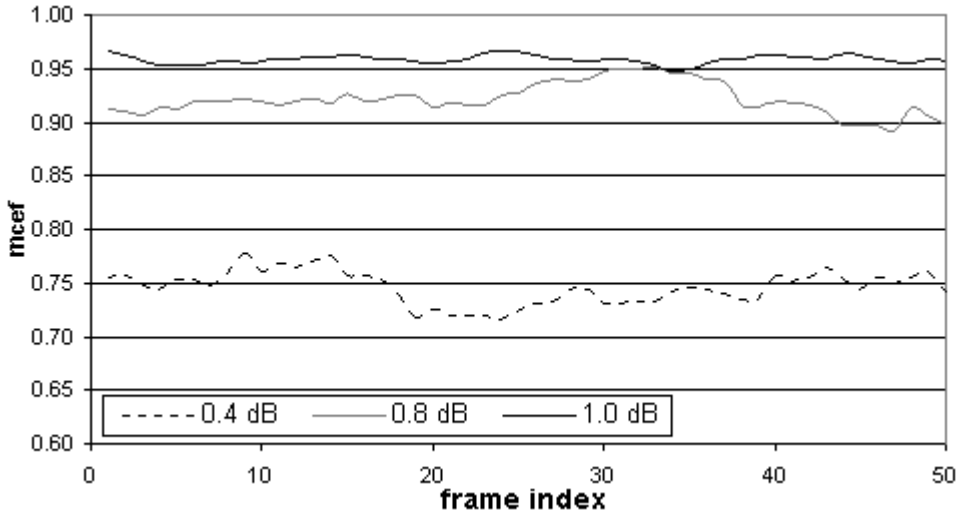


Figure 5.6 Metric at the output of a length 10 moving average filter

It should be noted that as the σ^2 for an information frame does not directly contribute to the calculation of the channel reliability factor, L_c, of the same frame, channel estimation errors cannot be avoided. However, considering the sensitivity of the turbo decoder error performance to channel misestimating, such estimation errors may not be significant under certain conditions [Summers & Wilson, 98] .

According to the previous findings (for a 1/3 rate code), within the 0.5-2.5 dB E_b/N_o range, a turbo decoder using log-MAP algorithm can operate with ±0.5 dB channel offset with no loss in BER performance. Therefore, in Figure 5.6, as long as the m_{cef} metric curve for an

average E_b/N_o value, S, does not intersect with the curves for S+0.5 and S-0.5 dB, the BER performance of the turbo code is not expected to degrade when the σ^2 estimate is obtained from m_{cef}.

Until now, the channel estimation problem mentioned in section 5.2, has been simplified to determining the relationship between the m_{cef} and the σ in the form of $f_\sigma(m_{cef})= \sigma$. The functional relationship, $f_\sigma(.)$, was determined by a practical approach with sufficient reliability, and will be explained next.

First, 60000 information frames were transmitted under various signal-to-noise ratios and were turbo decoded with perfect knowledge of σ. Then, by using the statistical data as a benchmark, a polynomial function was constructed to approximate the relationship between the m_{cef} and σ, for a given filter length. Finally, the previous two steps were repeated for different filter lengths.

Figure 5.7 shows the metric results for 4 different filter lengths; 5, 50, 100 and 500. It can clearly be seen that for all filter lengths, the m_{cef}-σ relationship follows a similar trend that can be approximated by a first-degree function also included on the same plot, where σ_e denotes the estimated σ value. It should be noted that approximation errors could be minimised by using higher order polynomials, however from a complexity point of view, higher degree polynomials increase the computational complexity. This compromise in accuracy is not expected to have a significant impact on the estimation performance due to the sensitivity of the turbo decoder.

Recall that the turbo decoder is not sensitive to channel estimation errors, as long as such errors stay within the ±0.5 dB E_b/N_o range. The scale of the y-axis in Figure 5.7 displays σ values for E_b/N_o ratios that are between 0.2 and 1.2 dB. This indicates that the magnitude of the maximum polynomial approximation error (about 0.15 dB) is smaller than 0.5 dB at all times. Therefore, the linear approximation provides sufficient estimation accuracy at a small complexity cost.

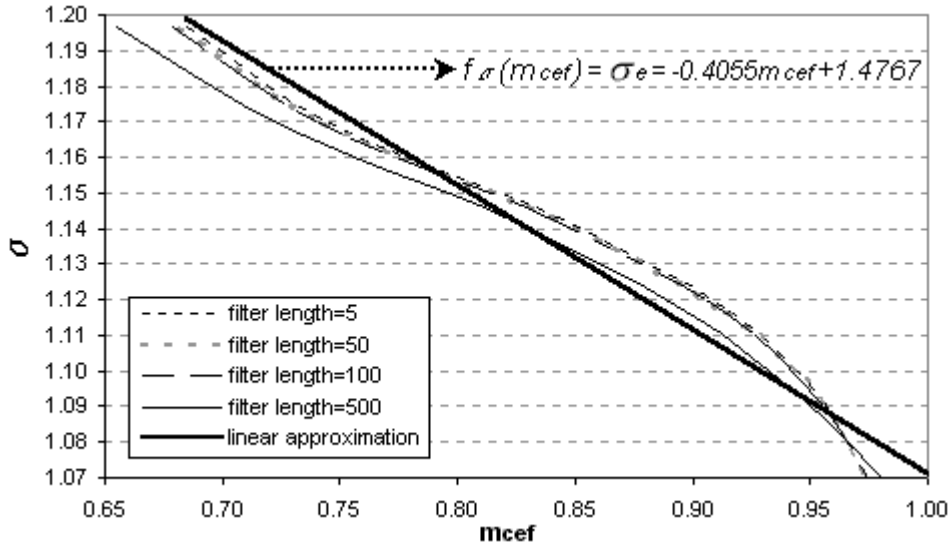


Figure 5.7 m_{cef}-σ relationship

Up to here, an initial knowledge of σ was assumed available at the decoder for determining the $f_\sigma(.)$ relationship. Even though, $f_\sigma(.)$ gives an acceptably accurate σ_e, a method for obtaining an initial σ_e is necessary for the estimation technique to work. A simple σ initialisation mechanism is explained in the next section.

5.6. Channel Initialisation

In the presence of additive white Gaussian noise, when an information symbol, x_k, is transmitted, a normally distributed zero mean random noise component, n_k, is added to it, whereby x_k is transformed into a noisy received symbol, r_k as in (5.7). From the error correction point of view, the objective is to uniquely identify the x_k, from the given r_k.

$$r_k = x_k + n_k \qquad \textbf{(5.7)}$$

As neither the n_k nor x_k are known at the receiver, none of them can be determined with a simple subtraction. If we are interested in obtaining information about the channel statistics, i.e. the σ, the exact values of a sufficiently large sample of n_k values needs to be known. A simple and straightforward solution to this problem is to uncover the x_k values, which is achieved by sending a set of x_k values, $\{x_k\}$, (often referred as the 'pilot symbols') whose values are known at the receiver. In this case, n_k values can uniquely be calculated from (5.7). Then, the σ of the noise components, which is expected to be close to the actual channel σ for sufficiently large $\{n_k\}$, can be calculated. The main disadvantage of sending a large set of pilot symbols is the reduction in information throughput, which results in inefficient use of the available bandwidth.

However, an alternative method of estimating σ by using the average magnitude of $\{n_k\}$, can minimise the above mentioned bandwidth inefficiency. Let us first consider how the average magnitude of $\{n_k\}$ are related to the σ of the channel noise.

Figure 5.8 shows the probability density functions (*pdfs*) of a normally distributed zero mean random variable, n, for 3 different variances (σ^2), 1.0, 2.25, and 5.29. In addition, on each *pdf* curve, two n_i points have been indicated such that (5.8) is satisfied. $P(-n_i \le n_k \le n_i)$ denotes the probability of n_k within the $[-n_i,n_i]$ range. In other words, the areas under the *pdf* curves within the specified $[-n_i,n_i]$ range are equal.

$$P(-n_1 \le n_k \le n_1) = P(-n_2 \le n_k \le n_2) = P(-n_3 \le n_k \le n_3) \qquad \textbf{(5.8)}$$

It should also be noted that as the σ^2 increases, the $[-n_i,n_i]$ range expands, as illustrated in Figure 5.8. This means that *increasing the σ, increases the probability of obtaining $\{n_k\}$ with high magnitudes*. Therefore, when the σ value of the channel increases (i.e. when the channel noise increases), the average magnitude of the noise components at the receiver also increases. If the receiver knows the relationship between the average noise magnitude, $|n|$, and the channel σ, it can easily estimate the σ using a small sample of n_k values.

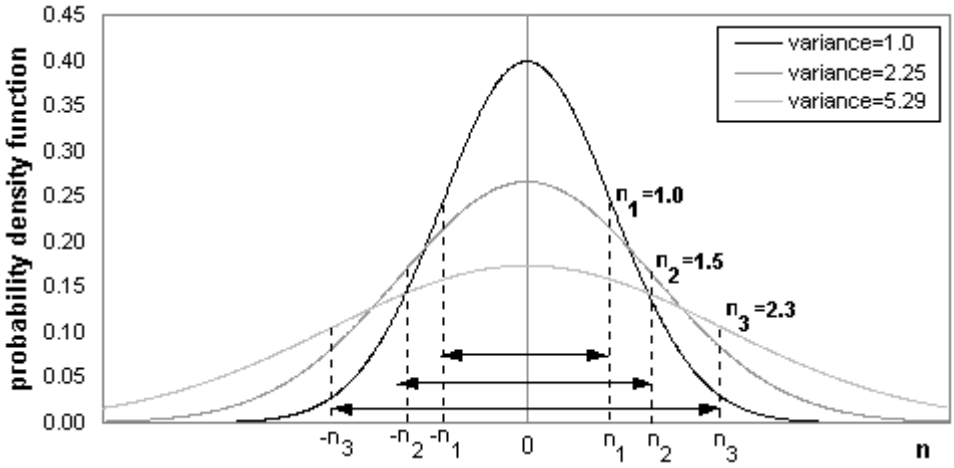


Figure 5.8 Probability density functions of normally distributed samples

Similar to the practical approach explained in section 5.5, a functional relationship between $|n|$ and the σ will be defined next. This time, 6000 frames, each with 9216 pilot symbols, are transmitted within the 0.2-1.2 dB range. The $|n|$ values for each simulated channel σ are calculated and plotted as shown in Figure 5.9.

Like $f_\sigma(.)$, the channel initialisation function, $f_n(.)$, unveils the first-degree relationship between $|n|$ and the σ. The accuracy of the relationship is also demonstrated with the closeness of the $f_n(.)$ and the standard deviation lines.

The channel estimation and initialisation algorithms explained so far can be considered as the constituent blocks of a channel estimator that can provide channel information to a turbo decoder. The following section describes the structure and the components of the proposed channel estimator as well as its operation.

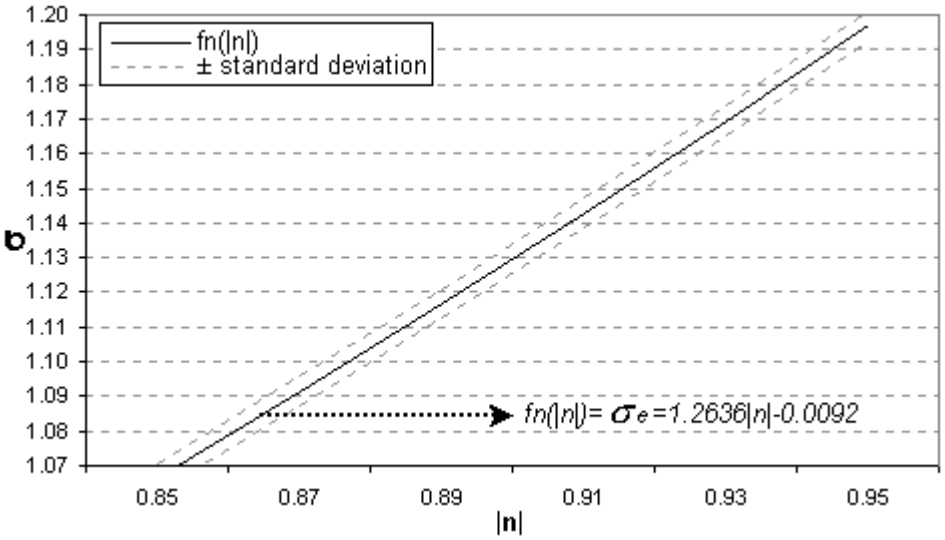


Figure 5.9 Average noise magnitude-σ relationship

5.7. Turbo Decoder with Channel Estimator

Figure 5.10 shows the combined channel estimator (encased with dashed lines) and turbo decoder blocks. The channel estimator consists of six sub-blocks, which are the channel initialiser, the moving average filter, the $f_\sigma(m_{cef})$ block, the progress word generators 1 and 2, and the compare-and-count blocks.

The channel initialiser predicts the channel conditions from the noisy received pilot symbol set, $\{r_k\}$. As $\{x_k\}$ is known beforehand, $\{n_k\}$ is simply extracted from $\{r_k\}$ differentially (see Figure 5.11). The average magnitude of $\{n_k\}$ is calculated and passed to the $f_n(.)$ function (see section 5.6) as an argument, whereby the initial σ estimate is obtained. The inverse of $f_\sigma(.)$ function (see section 5.5), denoted $f^{-1}{}_\sigma(.)$, accepts the σ value and outputs an initial m_{ce} value, denoted $m_{ce(init)}$, which is passed to the moving average filter. Note that the channel initialiser is switched off after generating a single output. From that point onwards, the remaining channel estimator blocks cooperatively predict the noise level by using the turbo decoder soft output.

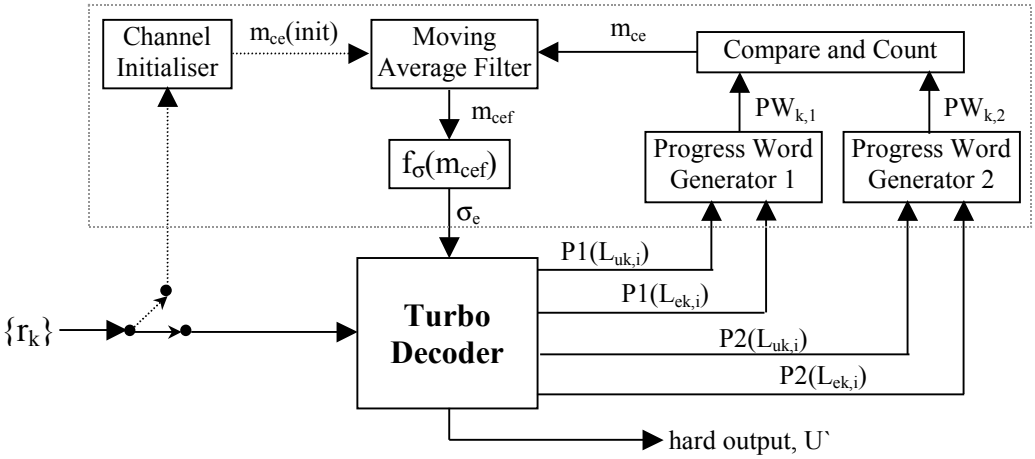


Figure 5.10 Turbo decoder with the soft channel estimator

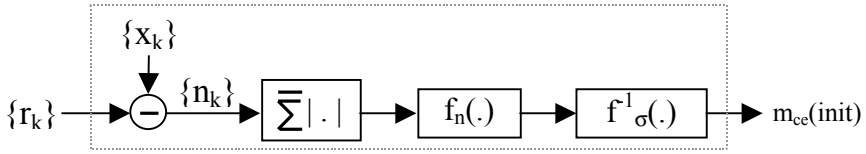


Figure 5.11 The channel initialiser

The moving average filter (Figure 5.12), also known as *the smoothing filter*, has *l* shift register elements (i.e. filter length=*l*), all of which are initialised by the m_{ce}*(init)*. Once the shift registers are set to m_{ce}*(init)*, the average is calculated, and delivered at the output as the estimation metric, m_{cef}. Clearly, the first m_{cef} will be the same as m_{ce}*(init)*, however, as the new m_{ce} values fill the registers, m_{cef} will vary according to the channel noise. Since m_{ce} is generated after decoding, channel estimation accuracy depends on the turbo decoder's soft value reliability.

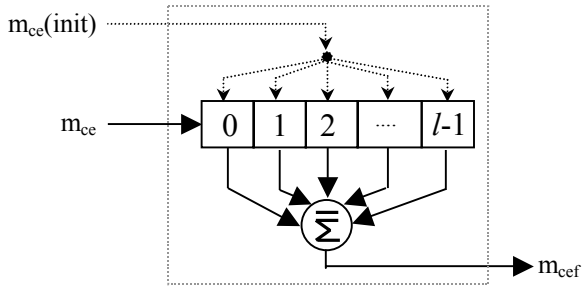


Figure 5.12 The moving average (smoothing) filter

The f_σ*(.)* block translates m_{cef} into σ_e, so that the turbo decoder can compute the channel reliability factor as in (5.4), after squaring the σ_e.

The turbo decoder is capable of delivering $L_{ek,i}$ and $L_{uk,j}$ for both parallel pipelines (denoted *P1* and *P2*) at its output, after each iteration. For each pipeline, one *PWG* block processes the magnitude progress of the extrinsic values for each information bit.

It should be pointed out that both *PWG* blocks shown in Figure 5.10, have identical internal structures, which was illustrated in Figure 5.3 earlier. The only modification in the *PWG* structure is the exclusion of the word weight counter. The operation of this counter is performed by the compare and count block.

The compare-and-count block accepts two PW_k inputs, each coming from a *PWG* block (Figure 5.13). Recall that in a *PWG* block *i*, one 8-bit codeword, $PW_{i,k}$, is generated for each information symbol, *k*. The weights of $PW_{1,k}$ and $PW_{2,k}$, are counted and added. As the codewords with weight 8 or more contribute to the estimation metric (see section 5.4), an *8+* counter block accumulates the number of such codewords (i.e. N_{8+}). After *N* codewords in an information frame have been processed by the compare and count block, N_{8+} is divided by *N*, whereby the estimation metric, m_{ce} is calculated. The new m_{ce} is passed to the smoothing filter for register update.

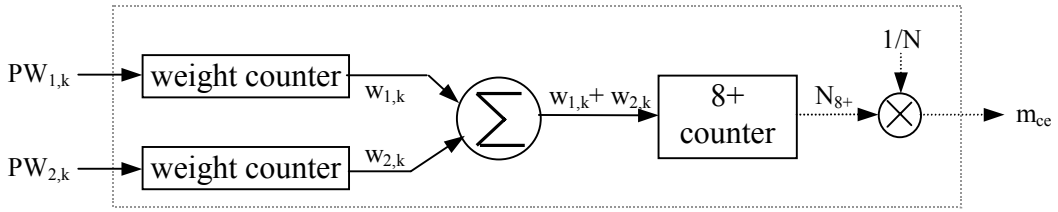


Figure 5.13 The compare-and-count block

Up to this point, the proposed channel estimation technique has been explained and the development of the technique into a channel estimator has been presented. In the next section, the performance results of the channel estimator are going to be presented.

5.8. Channel Estimator Performance

The performance of the channel estimator has been evaluated in three different categories. In each category four filter lengths, which are *l*=5, 50, 100 and 500, have been used.

In the first category, the accuracy of σ estimation is tested on a block basis. The σ tracking accuracy of the channel estimator is the second category, where σ is varied in the form of a sine wave, for three different frequencies. Finally, the BER performance of the turbo decoder with the channel estimator is compared to the ideal case, where the decoder has perfect knowledge of the channel σ.

Channel estimation accuracy has been tested for four different E_b/N_o ratios; 0.2, 0.6, 0.8 and 1.2 dB, as indicated in Figure 5.14 through Figure 5.17. The horizontal axis shows the

index of the frame, for which the corresponding σ estimate, denoted σ_e, is used. The channel σ values have also been marked in each figure.

It can be seen that for short filter lengths, locally, σ_e fluctuates more frequently from frame to frame compared to longer lengths. This can clearly be seen in Figure 5.14, for lengths 5 and 500. The stability of the σ_e, due to the improved smoothing properties of the moving average filter with long l values, is clearly noticeable. As the channel noise decreases, estimation stability for short l values is also achieved. One clear example to the last statement is displayed in Figure 5.17, where σ_e fluctuations are quite small for l=5. The reason for reduced σ_e fluctuations for short l values is the statistical nature of the channel noise. For a fixed probability, when σ decreases, the noise sample range also decreases (see Figure 5.8). Consequently, even by considering fewer noise samples, it is possible to obtain an accurate estimation of the channel σ. Therefore, the short-term history of σ_e (i.e. small l values) becomes sufficient to predict the current σ, as it gets smaller.

As the difference between the σ_e and σ decreases, the estimation accuracy is said to increase. When results presented in Figure 5.14 through Figure 5.17 are revisited, the estimation accuracy is observed to vary with the changing channel conditions. This can be explained by the curve approximation errors introduced by the $f_\sigma(.)$ function (Figure 5.7). For each filter length, the corresponding m_{cef}-σ_e relationship can be represented by a different function. However, for an acceptable trade-off in accuracy, one linear function can be used for all four filter lengths. The errors of such approximation appear as either over or under estimation of σ.

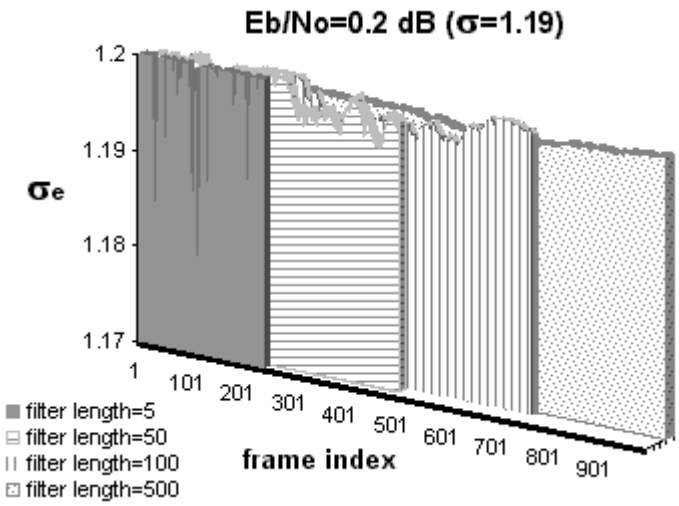


Figure 5.14 Sigma estimation accuracy at 0.2 dB

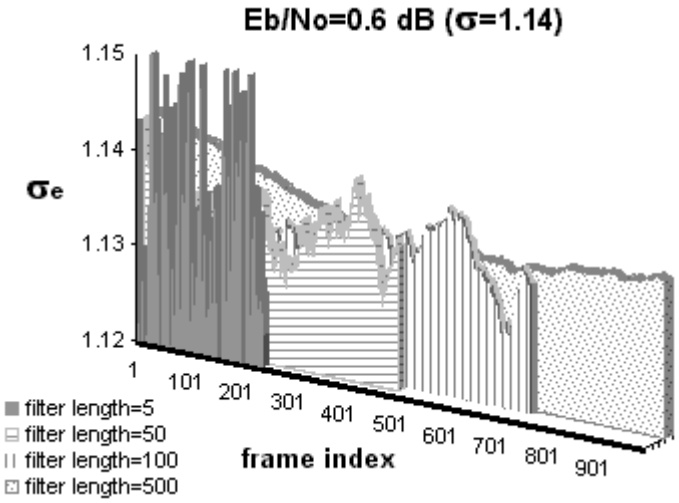


Figure 5.15 Sigma estimation accuracy at 0.6 dB

Let us consider Figure 5.16, where σ=1.11. For all filter lengths, on average, the σ_e is always less than σ, indicating an underestimation error, which is consistent with the data in Figure 5.7. Similarly, an example of overestimation can be observed in Figure 5.17, where σ_e is approximately equal to 1.08 whereas σ=1.07.

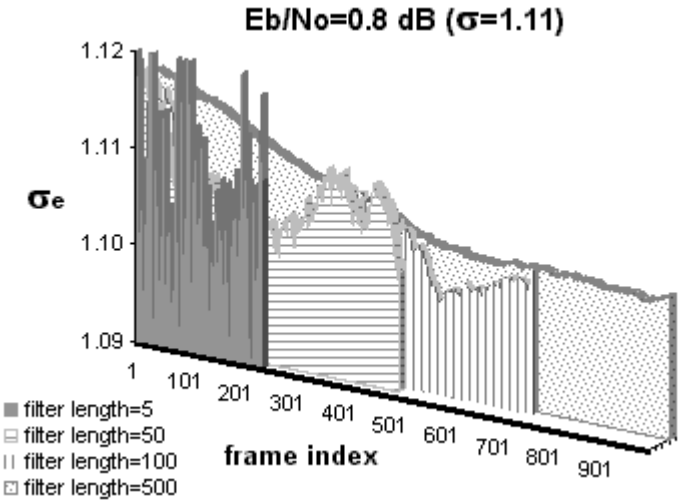


Figure 5.16 Sigma estimation accuracy at 0.8 dB

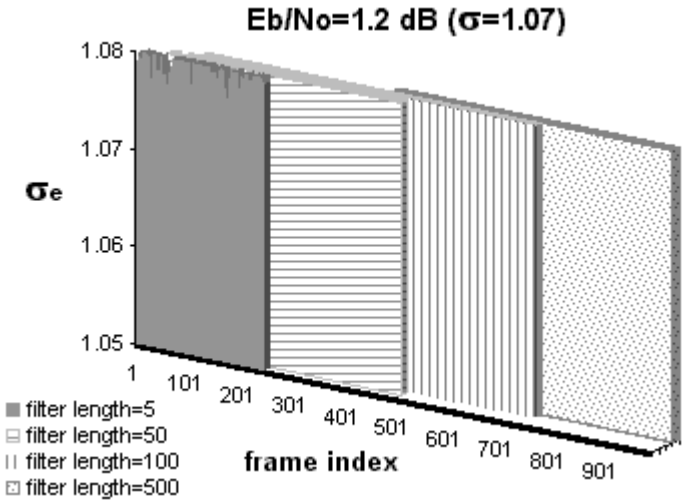


Figure 5.17 Sigma estimation accuracy at 1.2 dB

In order to test the σ tracking capability of the channel estimator, the σ values were forced to change from one block to the other, using a sine function with a 0.5 dB ($1.07 \leq \sigma \leq 1.19$) peak-to-peak amplitude. Three different frequencies (1000 frames/dB, 100 frames/dB, 5 frames/dB) were chosen to test the tracking capability using different filter lengths. Results are presented in Figure 5.18 through Figure 5.29.

The response time of the channel estimator increases with the increasing filter length for all σ frequencies tested. When the channel σ does not vary too fast from one block to the other, slow response of long filter lengths (e.g. l=500) generates tolerable errors in σ_e. However, when the channel variation frequency increases, slow response causes σ_e to drift away from σ, which in turn reduces the estimation accuracy. The reader is referred to Figure 5.21 and Figure 5.29 to observe the filter response and estimation accuracy relationship for l=500. Results indicate that the value of filter length needs to be chosen according to the rate at which the channel conditions change; *for fast changing channels small l is preferred*.

Even though minimizing the l values decreases the filter response time, and hence improves the σ tracking accuracy, the smoothing capability gets poorer and localized fluctuations of the σ_e become more pronounced. Typical examples of the last statement can be seen in Figure 5.18 to Figure 5.21.

When the σ variation amplitude is less than the sensitivity level of the turbo decoder to channel mismatch, such fluctuations do not affect the decoder performance. This sensitivity level is approximately ±0.5 dB E_b/N_o (see section 5.5) for the turbo decoder

used in our application. Therefore, localized fluctuations should stay within this sensitivity level in order not to degrade the decoder performance. In other words, t*he filter length should be optimised to give the fastest response, while keeping* σ_e *within the tolerable channel mismatch level*.

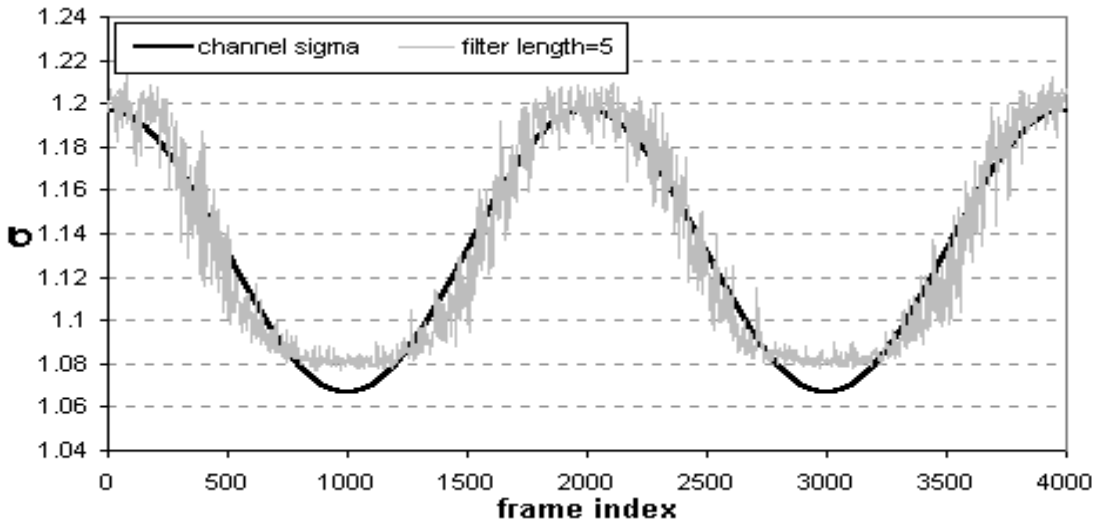


Figure 5.18 Sigma tracking for l=5 and σ change frequency of 1 dB/1000 frames

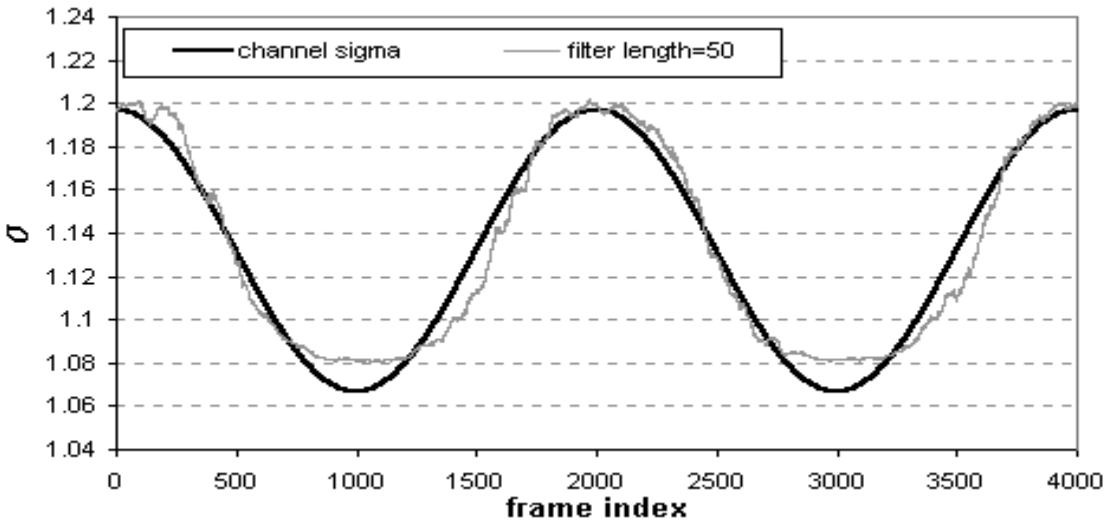


Figure 5.19 Sigma tracking for l=50 and σ change frequency of 1 dB/1000 frames

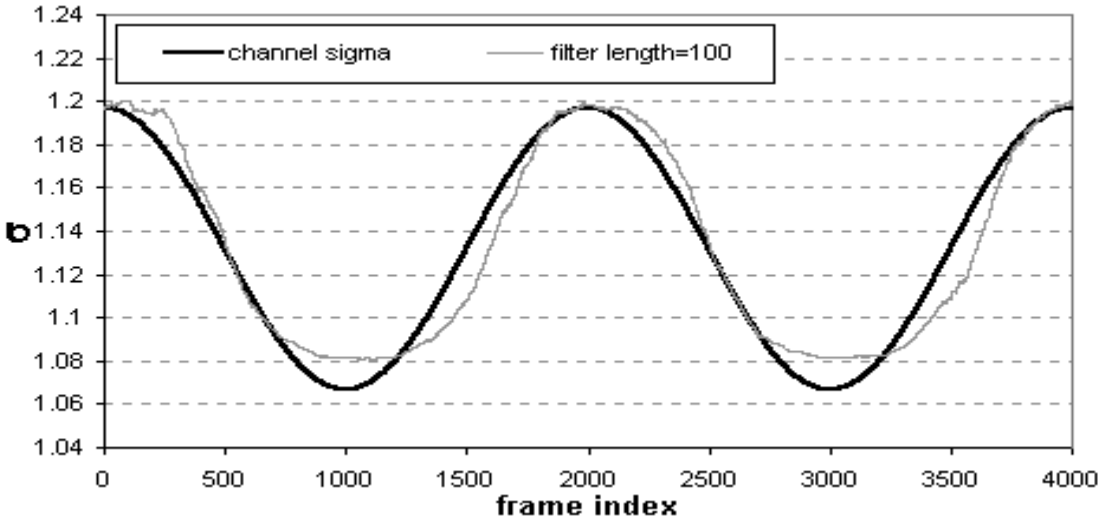


Figure 5.20 Sigma tracking for l=100 and σ change frequency of 1 dB/1000 frames

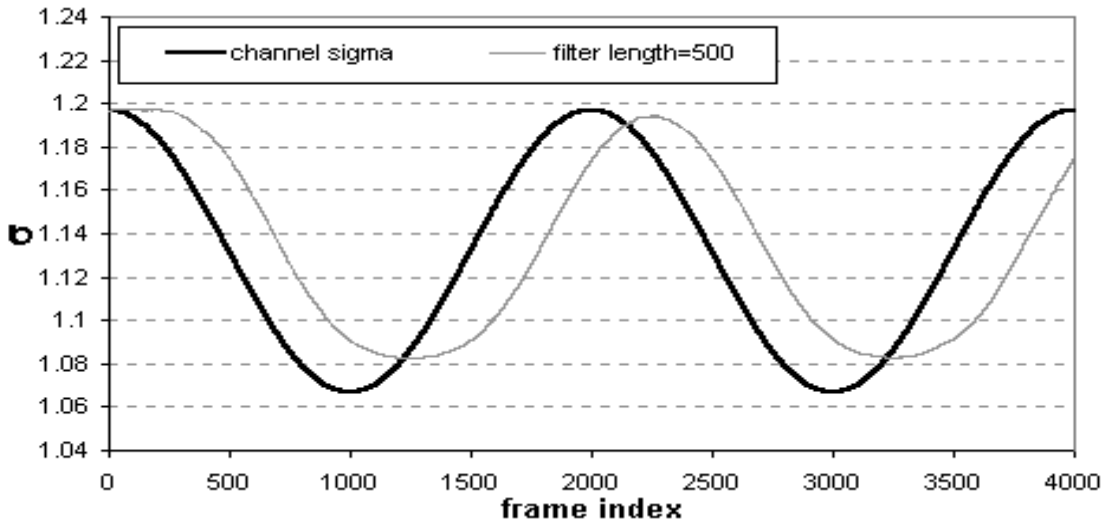


Figure 5.21 Sigma tracking for l=500 and σ change frequency of 1 dB/1000 frames

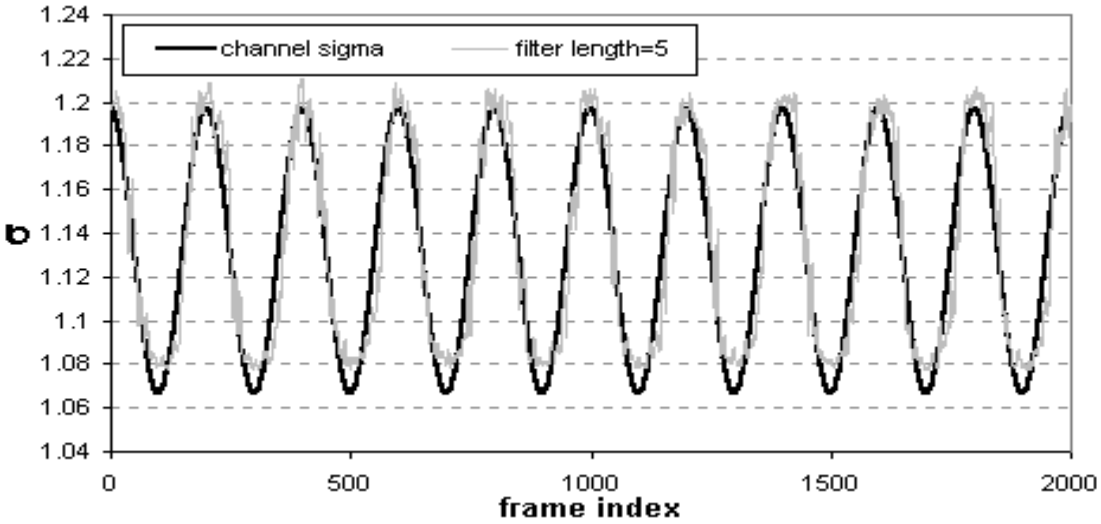


Figure 5.22 Sigma tracking for l=5 and σ change frequency of 1 dB/100 frames

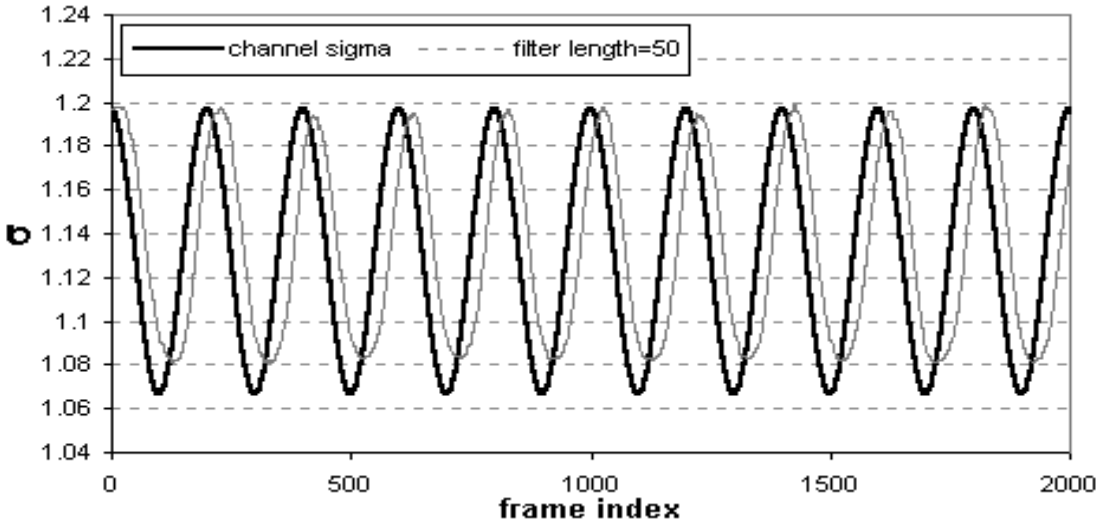


Figure 5.23 Sigma tracking for l=50 and σ change frequency of 1 dB/100 frames

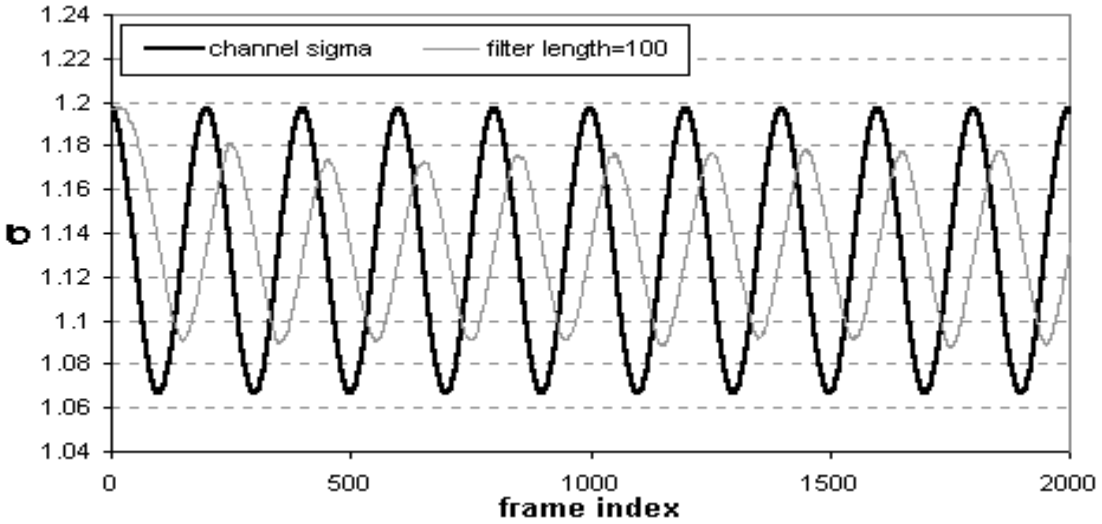


Figure 5.24 Sigma tracking for l=100 and σ change frequency of 1 dB/100 frames

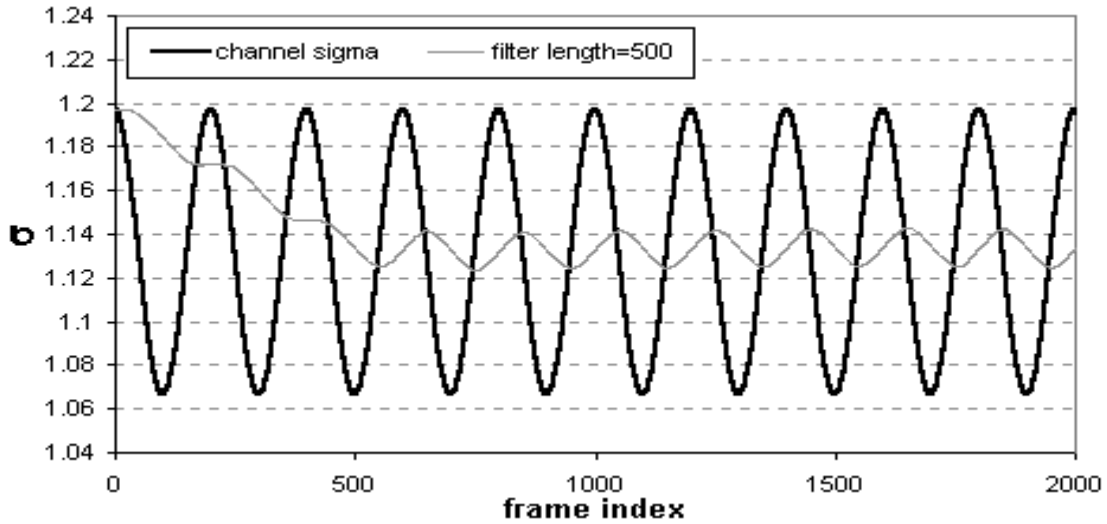


Figure 5.25 Sigma tracking for l=500 and σ change frequency of 1 dB/100 frames

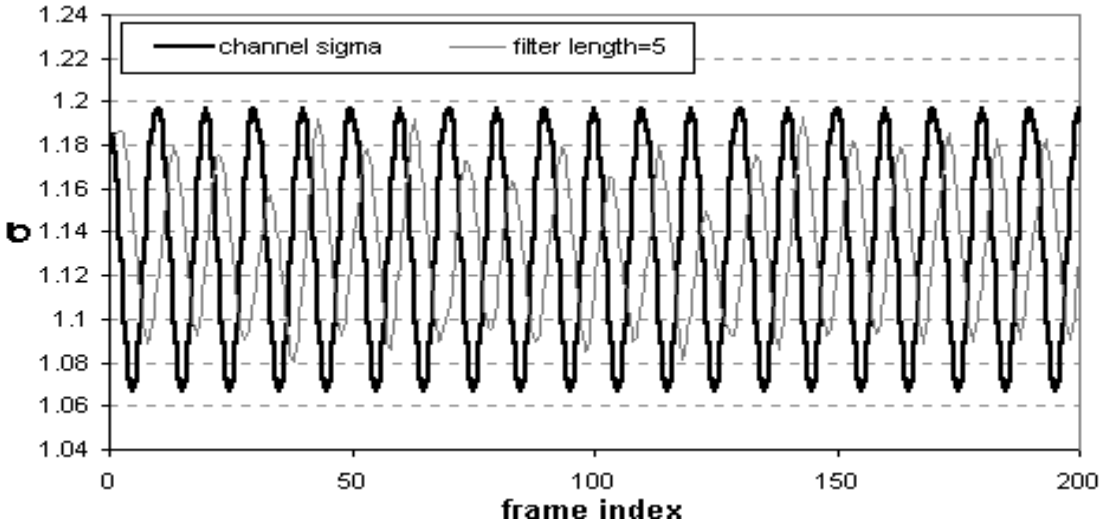


Figure 5.26 Sigma tracking for l=5 and σ change frequency of 1 dB/5 frames

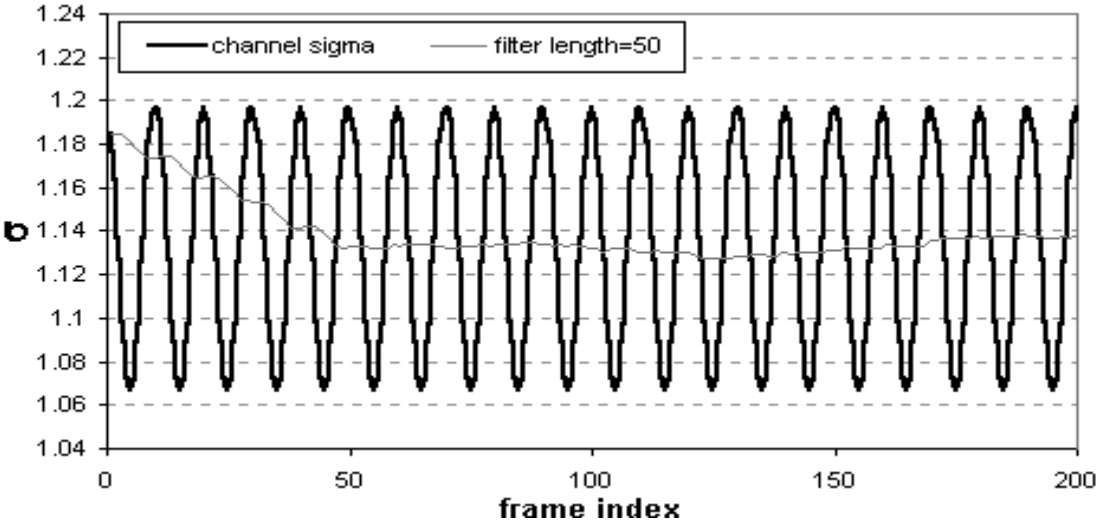


Figure 5.27 Sigma tracking for l=50 and σ change frequency of 1 dB/5 frames

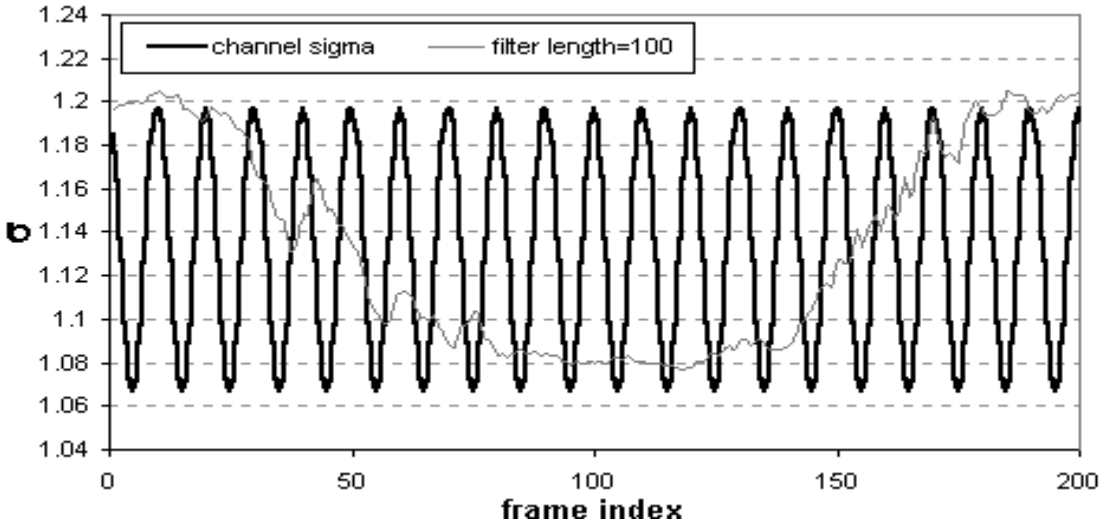


Figure 5.28 Sigma tracking for l=100 and σ change frequency of 1 dB/5 frames

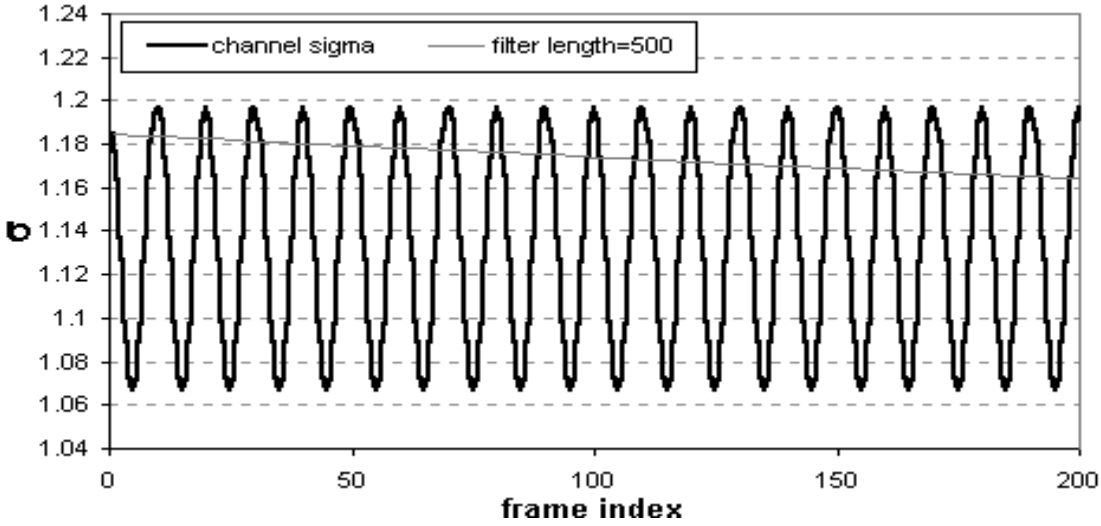


Figure 5.29 Sigma tracking for l=500 and σ change frequency of 1 dB/5 frames

The error performance of the turbo decoder with channel estimation was measured on both frame and bit basis. Bit and frame error rate plots are presented in Figure 5.30 and Figure 5.31, respectively. In each figure, error performance of the turbo decoder with channel estimation with four different filter lengths is compared to the case where perfect channel information is available (marked as *ideal performance*).

For all filter lengths, both the bit and frame error performance matches that of the ideal case, as expected.

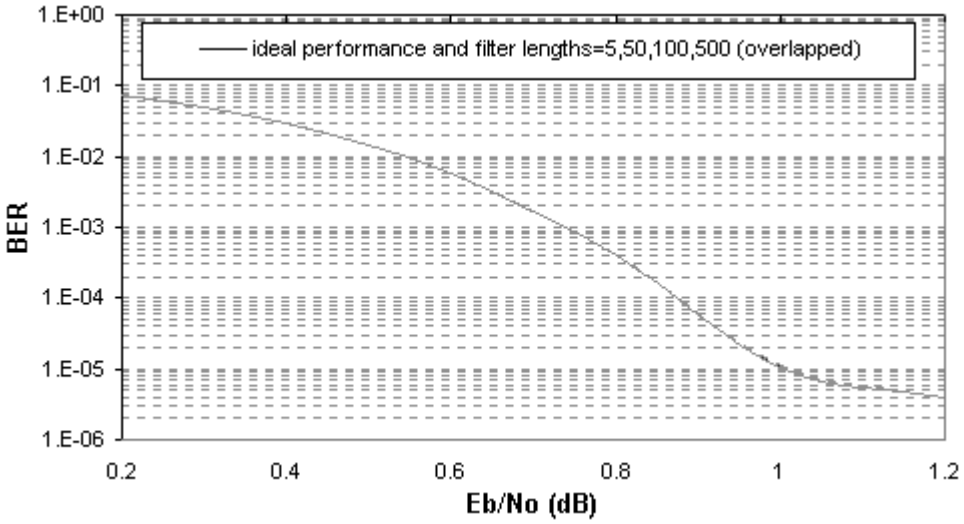


Figure 5.30 Bit error performance with channel estimation

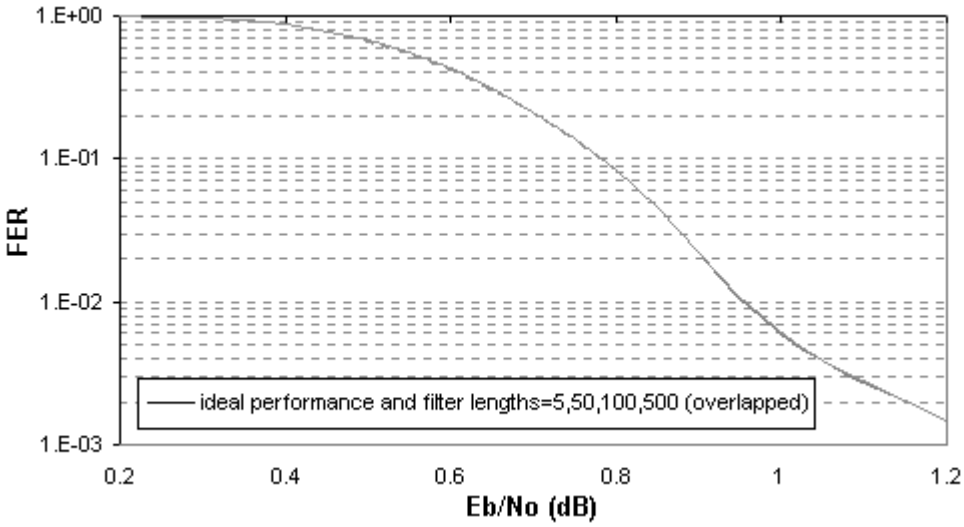


Figure 5.31 Frame error performance with channel estimation

5.9. Complexity Evaluation of the Channel Estimator

A similar complexity analysis to that described in Chapter 4 for the crossover detector will be used for the channel estimator as well. The processing times are expressed in units of one *and* operation, which is *T*.

First, the processing times for all operations performed in the channel estimator are going to be introduced. Then taking each sub-block of the estimator in turn, an operational complexity figure will be calculated. Thirdly, the complexity of the turbo decoder, which is

used with the estimator, will be calculated. Finally, the figures of the channel estimator and the turbo decoder will be compared.

Calculations performed by the channel estimator fall into seven different categories. They are bitwise shifts (*shift*), floating point shifts (*shift_flt*), floating point multiplications and additions (*fpo*), integer additions (*add_int*), integer comparisons (*comp_int*), comparisons of floating point numbers (*comp_flt*), and absolute value operations for floating point numbers (*abs_flt*). The delays intoduced by these operations are listed in Table 5.1. As it will be explained next, all the operations in the estimator can be modelled by various combinations of such operations.

The channel initialiser (Figure 5.11), performs 3N subtractions followed by 3N additions, 3N absolute value operations and one division. In the $f_n(.)$ and $f_\sigma(.)$ blocks a total of 2 multiplications and 2 additions are done. Since all the operations inside the initialiser accept floating-point arguments, the complexity for this block (C_{ci}) is expressed as in (5.9). Note that the division and subtraction operations are of type *fpo*.

Operation	Delay (T)
shift	1.4T
shift_flt	6.8T
fpo	7.1T
add_int	T
comp_int	6T
comp_flt	18.5T
abs_flt	22.6T

Table 5.1 Normalised operation delays for channel estimation

The moving average filter (Figure 5.12) performs l additions, 1 division and 1 shift per activation. Complexity figure for the filter (C_{maf}) is given by (5.10).

$$C_{ci}=(3\mathrm{N}+5)T_{fpo}+(3\mathrm{N})T_{abs\_flt}$$
$$C_{ci}=(89.1N+35.5)T \qquad \textbf{(5.9)}$$

The complexity of the $f_\sigma(.)$ block ($C_{f\sigma}$) is consists of one floating-point multiplication and one addition per activation (5.11).

$$C_{maf}=(l+1)T_{fpo}+T_{shift\_flt}$$
$$C_{maf}=(7.1l+13.9)T \quad \textbf{(5.10)}$$

For 8 iterations and N symbols, the *PWG* block makes I floating-point comparisons and performs $8N$ bitwise shifts during codeword construction. Equation (5.12) shows the processing complexity of one *PWG* block (C_{PWG})

$$C_{f\sigma}=2T_{fpo}=14.2T \quad \textbf{(5.11)}$$

The last channel estimator sub-block left to consider is the compare-and-count block (Figure 5.13). Inside each weight counter of this sub-block, (I-1) bitwise shifts and I integer additions are performed per information symbol. Following the weight count, N integer additions are carried out. The maximum number of integer additions that the *8+* counter can perform is N. One division concludes the calculation chain of the compare-and-count block. The overall complexity degree of this block ($C_{c\&c}$) is expressed in (5.13).

$$C_{PWG}=8T_{comp\_flt}+(8\mathrm{N})T_{shift}$$
$$C_{PWG}=(11.2\mathrm{N}+148)T \quad \textbf{(5.12)}$$

$$C_{c\&c}=7T_{shift}+(2\mathrm{N}+8)T_{add\_int}+T_{fpo}$$
$$C_{c\&c}=(2N+13.5)T \quad \textbf{(5.13)}$$

The turbo decoder complexity evaluation was explained in section 4.7. As the decoder used with the channel estimator is identical to the one in that section, the number of *fpo*s in (4.6) will stay the same. Note that even though the component decoders operate in parallel, this is not going to change the complexity, since our analysis is based on operational time delay rather than the total number of operations. The complexity of the turbo decoder (C_{dec}) can be expressed as in (5.14), in terms of T.

$$C_{dec}=256(N+1)T_{fpo}$$
$$C_{dec}=(1817.6N+256)T \quad \textbf{(5.14)}$$

Combining the expressions in (5.9) through (5.13), the channel estimator complexity (C_{chest}) can be obtained (5.15).

$$C_{chest}=C_{ci}+C_{maf}+C_{f\sigma}+C_{PWG}+C_{c\&c}$$
$$C_{chest}=(102.3N+7.1l+225.1)T \quad \textbf{(5.15)}$$

The C_{chest}/C_{dec} ratio allows us to compare the complexity of the channel estimator to that of the turbo decoder's. In most applications, l will be less than or equal to N. Therefore, for very large N values the comparative ratio will converge to approximately 0.06 (5.16).

$$\lim_{N\to\infty} \frac{C_{chest}}{C_{dec}} = \lim_{N\to\infty} \frac{109.4N + 225.1}{1817.6N + 256} \cong 0.06 \qquad \textbf{(5.16)}$$

Note that 0.06 is the highest value of the complexity ratio, as C_{ci}, is included in (5.16). However, since channel initialisation is performed only once prior to decoding, a more realistic complexity figure can be obtained by excluding C_{ci}. Recalculating (5.16) without C_{ci} gives us (5.17).

$$\lim_{N\to\infty} \frac{C_{chest}}{C_{dec}} = \lim_{N\to\infty} \frac{20.3N + 189.6}{1817.6N + 256} \cong 0.01 \qquad \textbf{(5.17)}$$

Regarding the complexity analysis presented above, it can be seen that the channel estimator initially increases the decoding delay by 6%. After the initial channel estimate is obtained, the channel initialiser becomes off-line, and thereafter the channel estimator increases the inherent turbo decoder delay by only 1%.

5.10. Discussion

In this chapter, a channel estimation technique that combines PSI and DDM has been introduced.

The new technique has been incorporated into a channel estimator block that performs channel initialisation as well as channel σ update, derived from the extrinsic information in iterative decoding.

Although each information frame is decoded by using the channel information obtained from the previous frames, the turbo decoder still performs as it would with perfect channel information. This indicates that the turbo decoder is almost unresponsive to channel estimation errors as long as they stay within the ±0.5 dB tolerance band.

It has also been shown that for AWGN channels, the channel estimation accuracy and stability depends on the filter length. In addition to that, due to the curve approximation errors, over and under estimation of the channel σ is inevitable. However, when the sensitivity of the turbo decoder to channel mismatch is considered, such estimation errors do not degrade the error performance of the decoder, under certain conditions.

Increasing the filter length in the channel estimator achieves stable σ estimation, at the price of longer response times, which can potentially degrade the error performance when the channel noise varies fast with high power. Therefore, the filter length should be chosen according to the rate and peak power of noise variations in the transmission channel.

In addition, the tracking capability of the new channel estimation technique has been tested assuming AWGN, by varying the signal-to-noise ratio using sine waves with different frequencies. It has been shown that the new estimation technique can follow channel variance deviations of different speeds, and therefore the same method can also be used for estimating fading channel conditions.

The cost of using the channel estimator described above, is, approximately, 6% increase in decoding delay during channel initialisation, and 1% at all times after initialisation, which is one of its advantages from an implementation point of view.

Chapter 6

Turbo Coded ARQ

6. Turbo Coded ARQ

6.1. Introduction

As a FEC scheme, many authors have verified the near-Shannon limit bit error performance of turbo codes since 1993 up to date [Berrou et al, 93] , [Hagenauer et al, 96] , [Massey & Costello, 00] , [TenBrink, 00] . Hybrid ARQ techniques, which combine turbo codes with error detection, are attractive options for obtaining low frame error rates besides low bit error rates.

The most straightforward error detection strategy in most of those hybrid ARQ systems has been to attach a CRC to each information frame, which is simple to implement and introduces a negligible decoding delay [Kim & Bahk, 96] , [Liu & Zarki, 97] , [Chan & Gerantiotis, 97] . Nevertheless, the detection accuracy of such systems is low without utilizing the turbo decoder's soft output information, particularly at low signal-to-noise ratios.

A few authors have managed to increase the error detection accuracy by using the turbo decoder's extrinsic information convergence patterns [Rasmussen & Wicker, 94] , [Buckley & Wicker, 00] . However, due to the computational intensity of such techniques, long decoding delays are inevitable.

A new error detection algorithm and a hybrid ARQ scheme [Coulton et al, 00] , that combines the two desirable features, namely the high detection accuracy and low processing complexity, is proposed.

In this chapter the detection technique, and its incorporation into a hybrid ARQ scheme with various retransmission strategies, is explained in detail. Section 6.2 describes the frame error detection metric, which is derived from the soft output of the turbo decoder. This detection technique is then incorporated into an ARQ system with 4 retransmission protocols, which is discussed in section 6.3. The detection accuracy, error performance and the throughput analysis of the hybrid ARQ scheme is presented in section 6.4. The final two sections in this chapter are the numerical complexity analysis and the discussion.

6.2. Detection Metric

Combining a powerful FEC scheme, such as turbo codes, with a frame error detection algorithm is an attractive solution, as it provides low FER as well as low BER. For this reason, a novel ARQ mechanism, which uses the soft information at the output of the decoder, has been designed to improve the FER performance of the turbo codes. During the development of the hybrid ARQ system, criteria such as accurate error detection, low pre-processing and decoding delay and high information throughput were taken into account.

It is known that the soft output of a turbo decoder gives both the hard value and the confidence level for an information symbol. The sign of the soft value determines the binary value (either 0 or 1) and the magnitude indicates how reliable the binary decoder decision is (high magnitude means high reliability). When this concept is extended to a group of information symbols (i.e. all symbols within an information frame), a frame reliability factor can be defined. In order to determine a measure for such reliability as a function of the decoder soft output, the relationship between the symbol soft values and the distribution of errors within a frame under various channel conditions needs to be brought into focus.

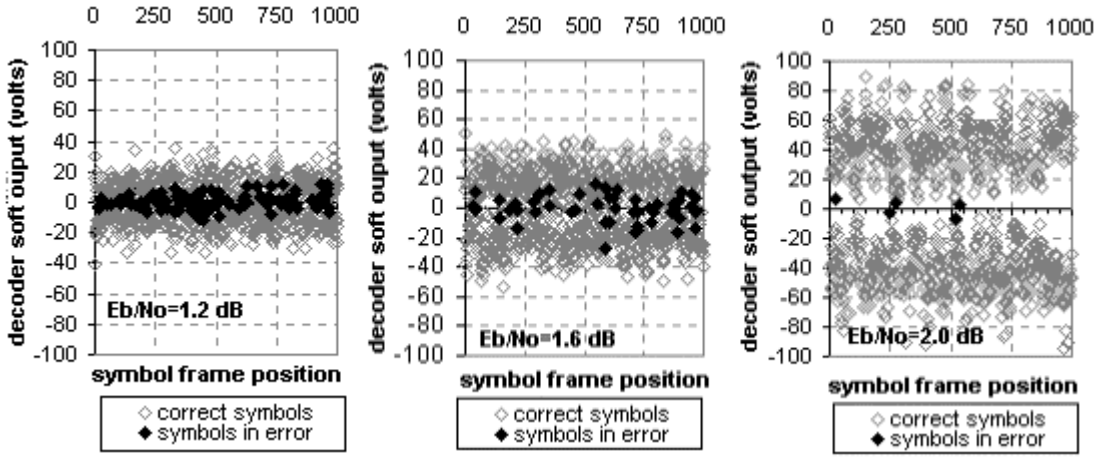


Figure 6.1 Decoder soft values under various channel conditions (frame size=1000)

Observations have revealed that the decoder soft values were distributed both in the positive and negative region with variable magnitudes, which are channel dependent (Figure 6.1). It should be noted that the confidence levels of the symbols in error are mostly smaller than the ones in the rest of a given frame. In order to pinpoint this comparison, Figure 6.2 is presented where the symbol energy within an information frame is visualized. Even though symbols in error appear to have low energy levels, low symbol

energy does not necessarily indicate an error. This fact is the reason why the frames with errors cannot be detected with perfect accuracy.

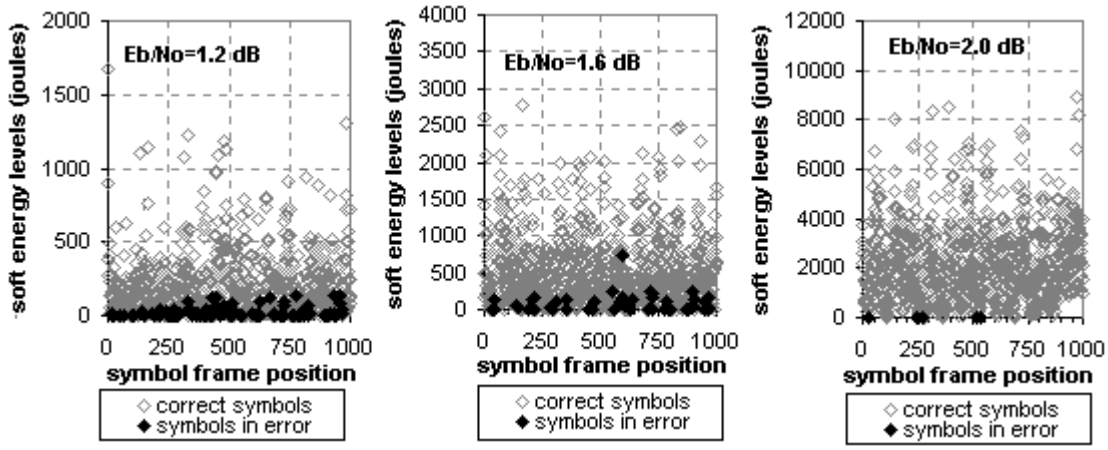


Figure 6.2 Decoder energy levels for various channel conditions (frame size=1000)

In an error detection process, the fundamental issue is the ability to differentiate frames in error. One effective tool for frame error detection can be the soft information energy levels. As will be explained, even though symbol energy levels are channel dependent, they provide sufficient reliability to be used for detection. In order to analyse the energy profiles of the frames with and without errors, 255 frames each with 1000 bits have been turbo encoded and transmitted over the AWGN channel under various conditions.

Turbo decoder soft output for all frames has been used in computing and displaying the corresponding energy levels (Figure 6.3). As the channel conditions improve, the average energy level per frame increases. The last statement holds true for all frames.

Another important insight gained from Figure 6.3 is that the average energy of the error-free frames is higher than that of the ones containing errors. The problem with using the energy difference as an error detection metric is that in all tested channel conditions the energy levels overlap. In other words, low energy levels may indicate an erroneous frame as well as an error-free one. Therefore, to determine the amount of energy overlap within a frame, analysis needs to be taken a step further.

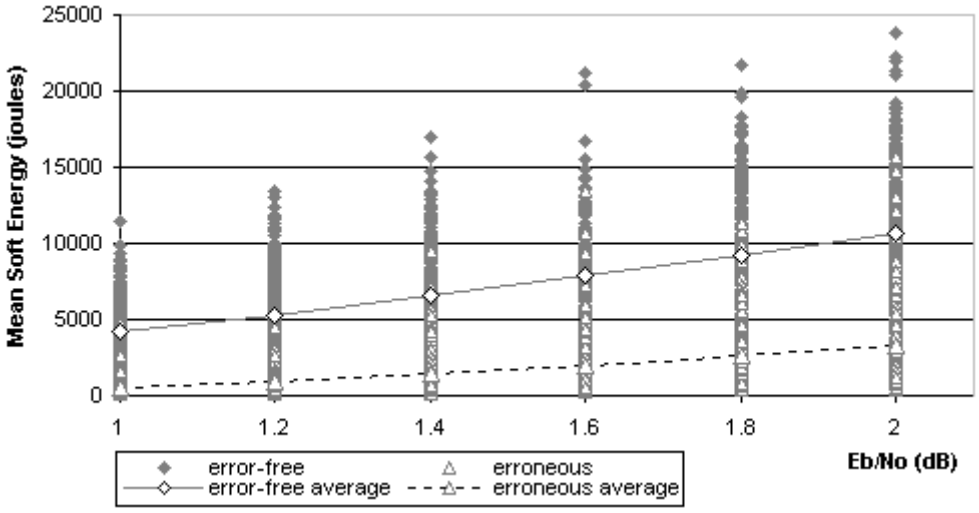


Figure 6.3 Energy profiles for erroneous and error-free frames (frame size=1000)

As a means of indicating the degree of energy overlap, standard deviations as well as the energy averages have been considered for 255 information frames (Figure 6.4). Despite the block-based observation presented in Figure 6.3, the energy profiles of the frames with and without errors do not overlap even within the tolerance bands introduced by the standard deviations. Put differently, the soft energy and its standard deviation can be used as a frame error detection metric in a hybrid ARQ system.

As far as the statistical analysis of soft energy is concerned, the standard deviation (σ) and averages (μ) appear to provide valuable information for differentiating between the frames in error and without errors, and therefore need to be combined somehow to define an error detection metric (Φ).

For a number of reasons, Φ is expressed as σ/μ, although other alternatives may be valid. In complexity terms, the advantage of defining Φ as a ratio is its simple and straightforward representation. Another advantage of Φ, as it will become clear later on, is its stability for large frame sizes as well as for improved channel conditions. This stability will be shown to help the error detection at the receiver.

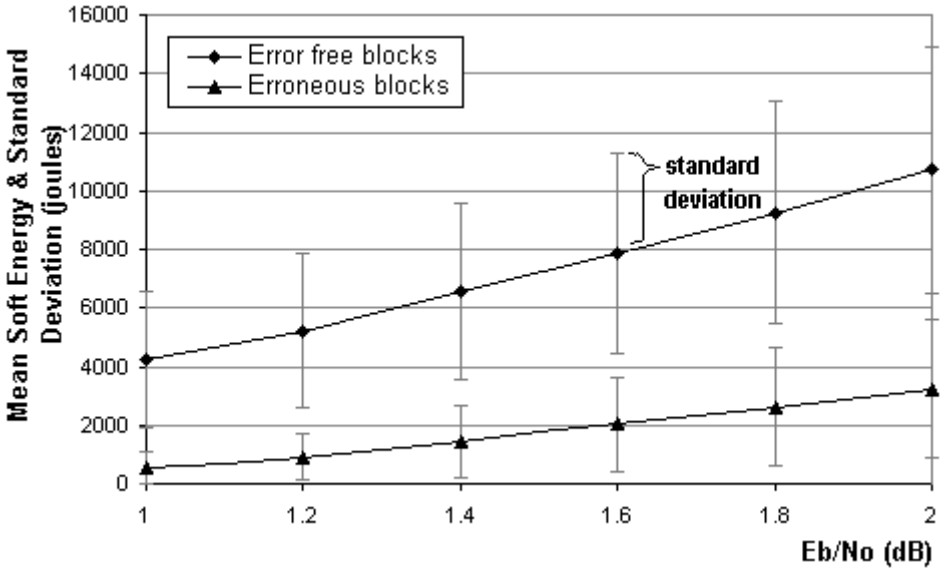


Figure 6.4 Soft energy profile of erroneous and error-free blocks (frame size=1000)

For further analysis, Φ for various frame lengths has been calculated and plotted for varying channel conditions (Figure 6.5). The Φ difference between the blocks containing errors and that without errors has been observed in order to determine how accurately the two frame types could be differentiated. The main criterion of accuracy for such a differentiation is the separation between their Φ curves for a given E_b/N_o ratio. Large distance between the two curves improves error detection, which means that the detection accuracy ranking of frame lengths would be 512, 1000, and 2000, in descending order.

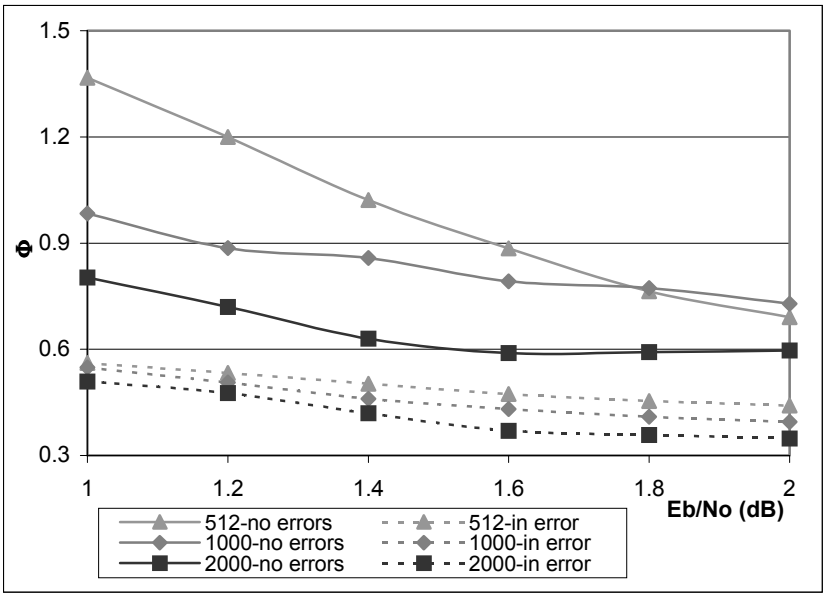


Figure 6.5 Φ detection ratio for 512, 1000 and 2000 frame lengths

Recall that Φ is the ratio of standard deviation to average. Even though the separation between the Φ curves is a measure of detection accuracy, revisiting the standard deviation-average relationship brings up a new perspective.

It can be argued that averaging is an approximated representation of a set of numerical samples and the standard deviation reflects the degree of accuracy of this approximation. As the difference between each sample and the average decreases, the standard deviation of all samples approaches to zero, which means that the approximation becomes more accurate.

When Figure 6.5 is reconsidered in terms of the reliability of averages, for error-free curves, Φ ratio gives us an important piece of information. As Φ increases (which means that the standard deviation diverges more from the average), the average reliability decreases. This observation becomes more distinct in the case of error-free frames, particularly for frame size of 512, around E_b/N_o=1.2 dB region.

Another important point that needs to be mentioned here is the fact that μ magnitude increases with improving channel conditions. Detection ratio Φ by itself, cannot indicate the magnitude of σ and μ separately. In other words, a small Φ does not necessarily mean a small σ, and consequently a reliable μ. A small Φ ratio can also correspond to large σ and μ values.

Knowing that μ increases with channel conditions (Figure 6.4), if Φ declines with the increasing E_b/N_o, it means that σ does not increase at the same rate as μ, which also indicates that sample fluctuations become less comparable to μ in terms of magnitude.

When Figure 6.5 is revisited, considering the decreasing trend of Φ, the essence of defining such a metric in terms of error detection reliability can be summarized with the following remark: "*Low reliability of averages for small frame sizes is counter balanced by the increased distance between the*Φ *ratios for error free and erroneous frames for accurate error detection. For large frame sizes, on the other hand, the reduced distance between the*Φ *ratios for the same frames is compensated by the increased reliability of averages to maintain accurate detection.*"

Even though Φ is suitable for error detection, it is channel dependent, which requires estimating the channel conditions prior to error detection. In order to find a way to avoid such estimation and the computational complexity it could introduce, relationships between the Φ ratios of blocks with and without errors (Φ_e, Φ_{ef}, respectively) were reconsidered. The difference between Φ_e and Φ_{ef} for various block sizes showed that Φ

could become channel independent with a simple approximation. Figure 6.6, a simplified version of Figure 6.5, illustrates two possible approaches to defining an error detection threshold. The threshold in the channel dependent model (Figure 6.6a) is equidistant to Φ_e and Φ_{ef} lines within the operating range. When the relationship between E_b/N_o and Φ is linear, threshold line is also linear. However, in reality this relationship is mathematically more sophisticated, which in turn makes it difficult to define a function for a detection threshold. Even after modelling such a relationship, yet an accurate channel estimation technique at the decoder is needed for the detection algorithm to work.

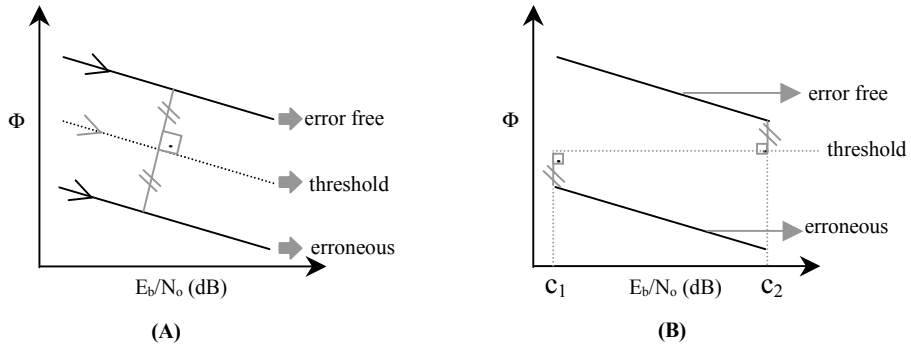


Figure 6.6 Error detection threshold models (A) channel dependent (B) channel independent

An alternative to the channel dependent model is presented in Figure 6.6b. According to this simplified model, a constant detection threshold (Φ_t) can be defined within the operation range of the decoder, as in (6.1). As long as the channel conditions stay within the operating range (i.e. $E_b/N_o \in [c_1, c_2]$), erroneous blocks can be detected by using Φ_t. Definition of a channel independent Φ_t might seem like an over simplification at first sight. However, Figure 6.5 justifies the use of such a threshold.

$$\Phi_t = \frac{\min(\Phi_e) + \max(\Phi_{ef})}{2} \quad \text{, such that } c_1 \le E_b / N_o \le c_2 \qquad \textbf{(6.1)}$$

As the channel noise decreases, the distance between Φ_e and Φ_{ef} starts becoming almost channel independent for all block sizes. More importantly, for large block sizes constant distance approximation is valid, even at low signal-to-noise ratios.

In capacity approaching turbo codes, large information block size is preferred as it is one of the factors that improve the decoder bit error performance. In other words, a constant Φ_t threshold is not only mathematically simple to define, but also is a detection metric that can be used in applicable turbo coding systems.

6.3. ARQ System and Protocol

So far the development of a detection metric, which uses turbo decoder soft information, has been explained in detail. In order to evaluate the detection accuracy, this novel detection algorithm was incorporated into a hybrid ARQ system. The overall structure of the ARQ mechanism, including the interconnection and function of constituent blocks and the retransmission protocols used, is explained in this section.

The block diagram of the hybrid ARQ system is given in Figure 6.7. The output of the turbo encoder is BPSK modulated. The modulated frame is transmitted over an AWGN channel and is demodulated at the receiving end, where soft information is fed to the soft-input/soft-output (SISO) decoder. The turbo decoder performs 8 iterations and passes the log-likelihood ratios (LLRs) of all information symbols to the frame error detector (FED) block (Figure 6.8).

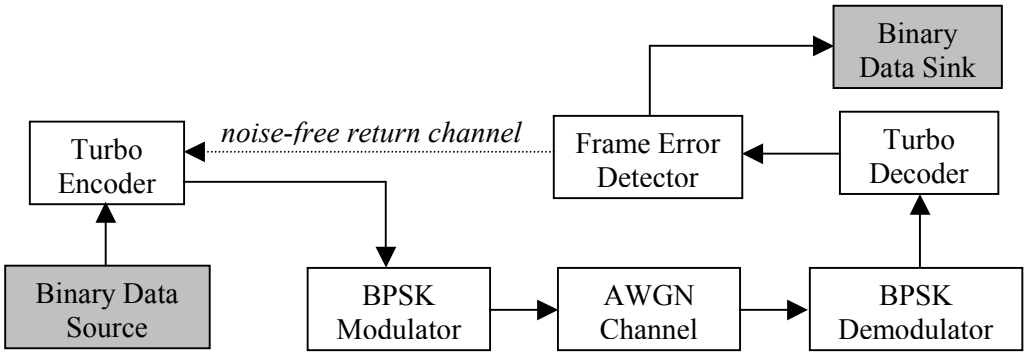


Figure 6.7 Hybrid ARQ System

The FED block processes the output of the turbo decoder in order to decide whether an ARQ is necessary. When no frame errors are detected, the FED block sends a *positive acknowledgement* (ACK) flag over the noise-free back channel[•] to the encoder. Based on the soft information from the decoder, FED block outputs all the hard values for the decoded information symbols.

If retransmission is needed, a *negative acknowledgement* (NACK) flag is sent to the encoder. Depending on the ARQ mode, as it will be explained shortly, the transmitter sends the required information to the receiver. In order to minimize the decoding delay per frame, the number of ARQ requests on any decoded information block is limited to one.

[•] Even though a noise-free back channel does not exist, it is assumed that sufficient error protection to ensure accurate reception of flag information is used over the AWGN back channel.

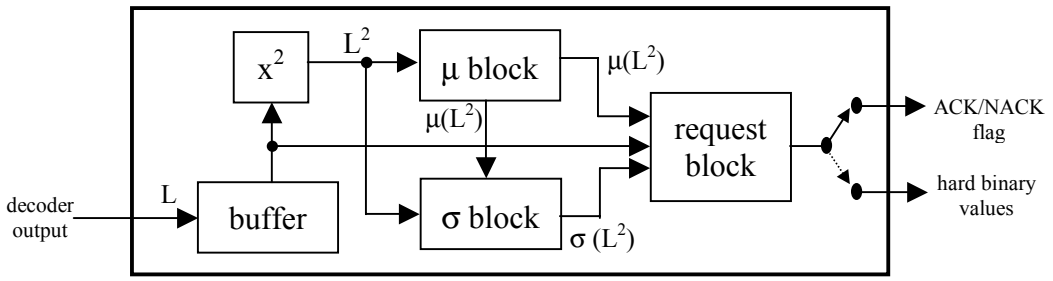


Figure 6.8 Frame Error Detector

The FED block consists of 5 internal blocks: soft value buffer, squaring, μ, σ and the request blocks (Figure 6.8). The buffer can be thought of as an array of frame size that stores the LLR ratios of each information symbol, provided by the turbo decoder. The squaring block simply computes the soft value energy levels for the decoder soft outputs. The μ and σ blocks calculate the average and the standard deviation, respectively, for the soft energy levels. The request block calculates the Φ ratio and compares it to the relevant Φ_t, as described in section 6.2, and either outputs an ACK or NACK flag. It also outputs the hard values for the decoded information frame, when ACK flag is generated.

Prior to a retransmission request, information bits and the unpunctured parity bits from the first and second component encoders are transmitted together with the flush bits over the AWGN channel. After the FED block generates a NACK flag, depending on the operation mode, different types of additional information are retransmitted, to correct more errors in the current information frame. Four different modes of operation have been defined in the proposed hybrid ARQ system (Figure 6.9). Differences between these modes are the amount and type of information requested by the receiver, for a detected frame in error. Details of each ARQ mode will be explained to give a clearer picture of the operation principle. In the rest of this chapter, information packet transmitted before and after retransmission will be referred to as the *primary* and *secondary*, respectively. The four ARQ retransmission modes operate as follows.

MODE 0: *The **punctured** parity bits from **either** the first or the second component encoders* are transmitted (in our application punctured parity bits from the first encoder are sent). In this mode, the primary information is not discarded.
MODE 1: *The **punctured** parity bits from **both** the first and the second component encoders* are transmitted. As in mode 0, the primary information is not discarded.
MODE 2: *The **unpunctured** parity bits from **both** the first and the second component encoders* are retransmitted. The information symbols in the primary information packet are

kept, however the parity information is replaced with that in the secondary information packet.

MODE 3: *The **primary information** packet* is retransmitted as the secondary information. Primary information is discarded in this mode.

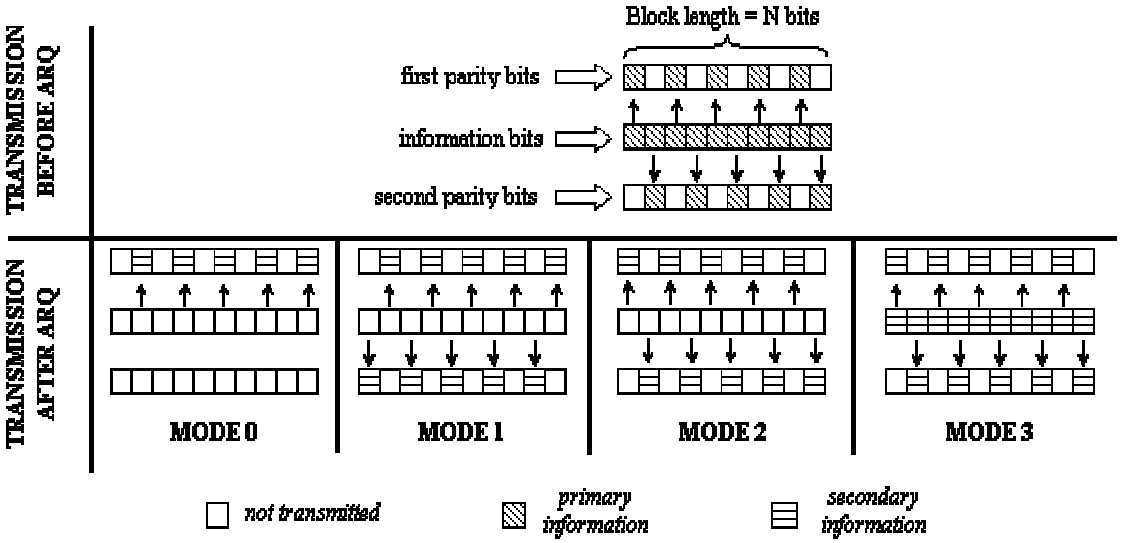


Figure 6.9 ARQ Operation Modes

Modes 0, 1, and 2 are type-II hybrid ARQ schemes since received symbols are kept for later use in decoding. However, mode 3 is a member of type-I hybrid ARQ class, as the information and the parity symbols are discarded and requested from the transmitter when a frame is detected in error.

The amount of secondary information transmitted reduces the overall code rate, and consequently the information throughput. When no retransmission is necessary, the code rate (R) is ½. Depending on the probability of retransmission (P_A) and the operation mode, the effective code rate (R_{eff}) can vary. In this section the calculation of R_{eff} for mode 0 will be explained as an example. R_{eff} for all other ARQ modes can be derived in the same way.

$$R = \frac{k}{n} = \frac{k}{k+k} = \frac{1}{2} \qquad \textbf{(6.2)}$$

Primary information is transmitted for each information block, and consists of k information bits and k parity bits. In other words, for k information bits, a total of $n = 2k$ bits are transmitted. The code rate, in this case, can be calculated as in (6.2).

Transmission of the secondary information increases the number of redundancy bits per k information bits. For example, in mode 0, when the FED block generates a NACK flag with a probability P_A, an additional $k/2$ parity bits are sent to the receiver. In other words,

the amount of redundant information increases by $k/2$ with a probability of P_A. Therefore, the effect of additional ARQ overhead in mode 0 on R can be expressed as in (6.3).

$$R_{eff} = \frac{k}{2k + (\frac{k}{2} P_A)} = \frac{2}{4 + P_A} \qquad \textbf{(6.3)}$$

In order to determine R_{eff} for a given mode of operation, P_A needs to be known. It can be calculated empirically by dividing the number of NACK blocks to the total number of information blocks transmitted. It should also be noted that P_A is a channel dependent ratio. Deteriorating channel conditions increase the bit error rate of the turbo decoder and consequently, the frequency of repeat requests ascends. As a result, P_A approaches 1 and R_{eff} decreases.

Highest throughput can be achieved at the highest code rate (i.e. R), which is when P_A is 0. Performance of each ARQ mode in terms of information throughput can be evaluated simply by using R_{eff} and P_A. Table 6.1 summarizes the relationship between R_{eff} and P_A, for each ARQ mode as well as the maximum and minimum possible code rates (R_{max} and R_{min}) in each case.

The next section presents the error and the throughput performance of the hybrid ARQ scheme explained so far.

	Retransmission overhead	Effective code rate (R_{eff})	R_{max} (P_A=0)	R_{min} (P_A=1)
MODE 0	$(k/2) \cdot P_A$	$\frac{2}{4 + P_A}$	1/2	2/5
MODE 1	$k \cdot P_A$	$\frac{1}{2 + P_A}$	1/2	1/3
MODE 2	$k \cdot P_A$	$\frac{1}{2 + P_A}$	1/2	1/3
MODE 3	$2k \cdot P_A$	$\frac{1}{2 \cdot (1 + P_A)}$	1/2	1/4

Table 6.1 R_{eff} for all ARQ modes

6.4. Hybrid ARQ Scheme Performance

The FEC used in the hybrid ARQ system is RSC turbo code with rate ½. Decoder, which operates in serial mode, performs 8 iterations before an ARQ is triggered. The number of retransmissions per block is limited to one, in order to minimise decoding delay. Three different information frame sizes (512, 1000, and 2000 bits) have been simulated over

AWGN channel under different noise levels. The transmitter can be configured to use one of the four ARQ modes (either mode 0, mode 1, mode 2 or mode 3) at a time, in response to the receiver's repeat requests.

Detection accuracy, information throughput and FER performance are evaluated for four modes and three frame sizes, and the results are presented in the rest of this section.

6.4.1. Frame Error Detection and Correction

Detection algorithm, as explained in section 6.2, requires a comparison of the Φ metric with Φ_t that is specific to a frame size. Then the FED block generates either an ACK or a NACK flag, depending on the result of the comparison. Once an information frame is decoded with the hybrid ARQ system, there are four possible outcomes, as summarized in Table 6.2.

In the first case, FED block decides that the frame has no errors and therefore generates an ACK flag when the frame is actually error free. In this case, FEC attempts to correct all frame errors by itself. FED block can also generate an NACK flag once it correctly detects a frame in error, as in case 2. The first two cases are presented as examples of the expected operation of a detection algorithm.

CASE INDEX	*FED FLAG*	*FRAME ERROR STATE*
1	ACK (no ARQ)	no errors
2	NACK (ARQ)	errors
3	ACK (no ARQ)	errors
4	NACK (ARQ)	no errors

Table 6.2 All possible outcomes of FED decisions

However, there might be instances when detection can fail in two ways (cases 3 and 4 in Table 6.2). FED block can either miss a frame in error and generate an ACK flag or it incorrectly detects a frame as in error and sends an NACK flag.

Figure 6.10 through Figure 6.12 presents the detection accuracy performance of the proposed hybrid ARQ system, for three different frame sizes. Cases 1 and 2 discussed above are marked as '*error free FEC*' and '*accurate detection*', respectively. The remaining two cases are combined and are denoted as '*missed/wrong detection*' on each plot.

As the increased frame size improves the BER in turbo codes, the ratio of retransmitted frames decreases for moderate and high E_b/N_o ratios. When a frame is severely affected by the channel noise, FEC starts performing poorly. That is, when the majority of blocks is detected to have errors and is flagged with a NACK by the FED block.

Recall that the detection metric, Φ, is dependent on the average soft energy of information frames. The energy level of a frame is inherently channel dependent as it is calculated from the a posteriori decoder soft values. Turbo decoder bit confidence levels increase with decreasing noise in the AWGN channel, which in turn increases the average energy levels for all frames.

It was explained in section 6.2 that the standard deviation of energy does not change as much as the average energy levels with varying channel conditions. Therefore, the average energy becomes the dominant factor in Φ term under less noisy conditions, and the detection accuracy starts to be determined mainly by the average energy term. This is the reason why the ratio of missed or wrongly detected frames decreases as the channel conditions improve.

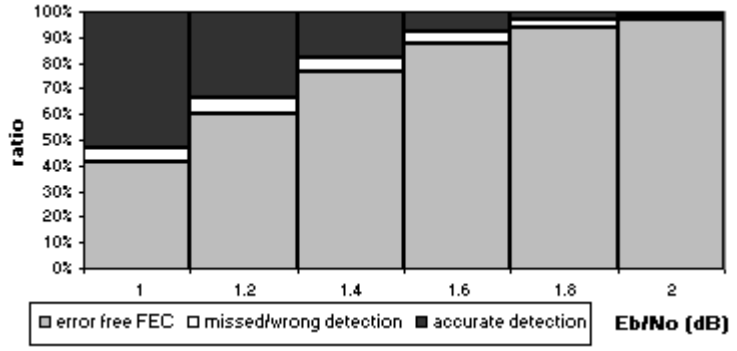


Figure 6.10 Detection accuracy for *N*=512

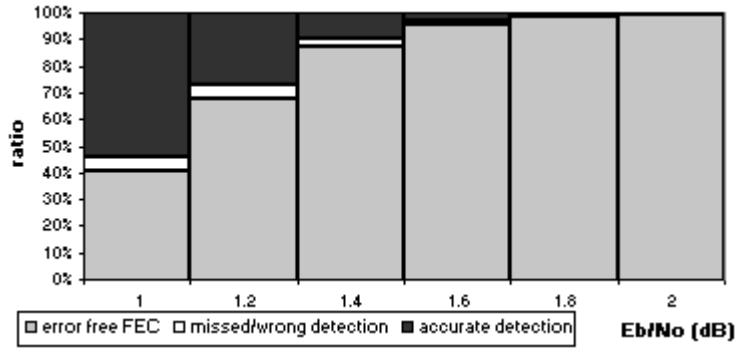


Figure 6.11 Detection accuracy for *N*=1000

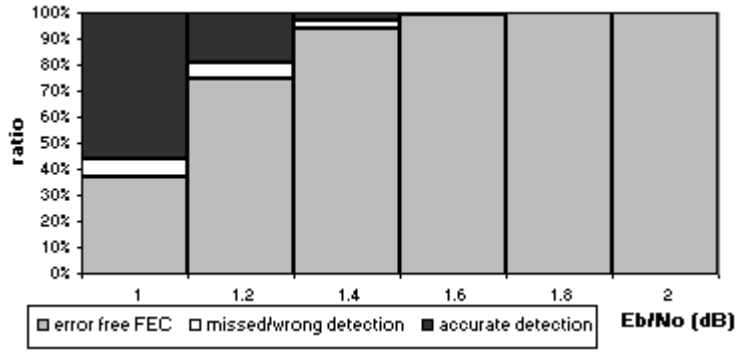


Figure 6.12 Detection accuracy for *N*=2000

FER performance for all ARQ modes and frame sizes has been presented in Figure 6.13 through Figure 6.15.

The lowest FER is achieved by mode 1, as it provides the highest level of protection for an information frame by sending the corresponding full parity sequence, following a NACK flag.

Mode 0 is the second best scheme in terms of FER. It is designed to send half of the redundancy of mode 1 in retransmission. Note that the performance difference between modes 0 and 1 becomes less significant with the increased frame size (i.e. increased interleaving degree).

Recall that the turbo decoder used in the ARQ system operates in serial mode, which means that the second component decoder uses the first decoder extrinsic information as its a priori information. In other words, in serial decoding soft information is generally biased towards the first component decoder output. This is the reason why the difference between mode 0 and mode 1 performance is small. Once the punctured parity bits are transmitted as in mode 0, iterative decoding reaches a high convergence level for the decoded frame. Even though the second punctured parity bits are also transmitted in mode 1, little improvement can be achieved by processing this overhead. That is to say, even though the second set of parity bits still improve the confidence levels, the hard value for most information bits is unlikely to change as a result.

When the interleaving degree is increased, the first component decoder passes more reliable extrinsic information to the second component decoder as the a priori input. The second decoder extrinsic output, in this case, is not large enough to change the hard value for the majority of information bits. In this case, the bit error rate that has already been reduced to a low level can only be marginally reduced further. Consequently, mode 0 and mode 1 perform almost the same in terms of FER. A good example to this is displayed in Figure 6.15.

As the channel conditions deteriorate, the coding gain introduced in the case of mode 1 exceeds that of mode 0. This is due to the decreased magnitude of soft information at the output of the first component decoder, which can be improved further by the second component decoder. As the second component decoder will generate extrinsic information based on a larger number of parity bits, reliability of soft information in this case will be significant. Figure 6.13 clearly illustrates the superiority of mode 1 over mode 0 at low E_b/N_o ratios (less than 1.4 dB). Under less noisy channel conditions the FER for both ARQ modes converge.

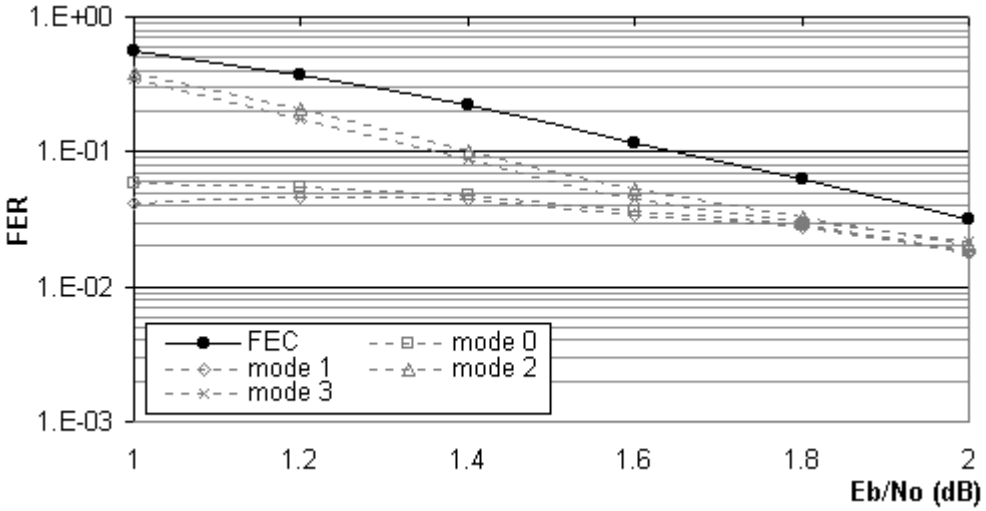


Figure 6.13 Frame error rate for *N*=512 bits

Modes 2 and 3 perform poorer than the first two modes. Secondary information transmitted in modes 2 and 3 is a repetition of all or a part of the primary information in a different time slot. Retransmission in a different time slot can reduce the amount of noise that affects the modulated symbols. Therefore, if the majority of bit errors in a frame could not be corrected by the turbo decoder due to high level of noise at the time of transmission, repetition of information can help eliminate most of those errors. However, if the channel is too noisy, repetition will be of little use, as a similar level of noise that has corrupted the primary information will also affect the secondary information.

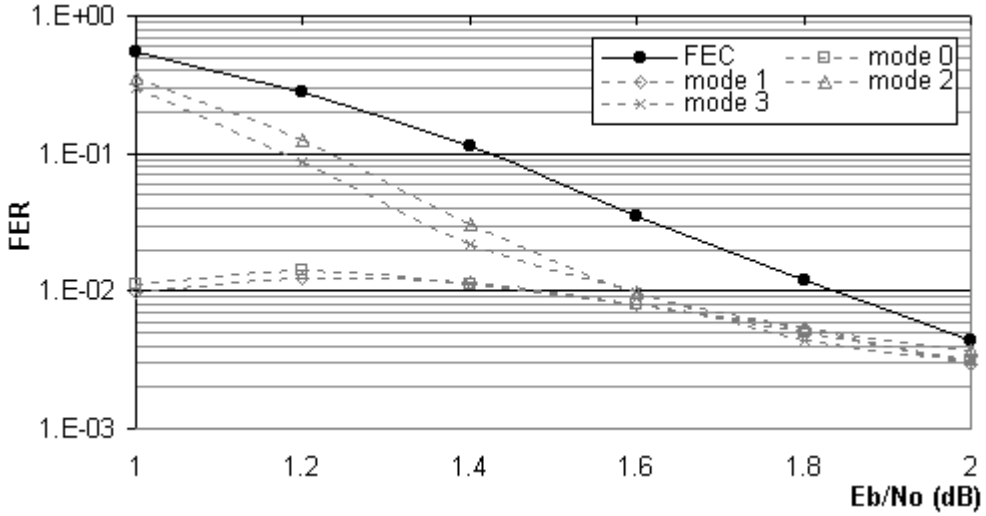


Figure 6.14 Frame error rate for *N*=1000 bits

Figure 6.13 through Figure 6.15 illustrate the performance of modes 2 and 3 for various frame sizes. Note that the coding gain provided by the two modes increases as the channel conditions improve. FER performance of mode 3 is slightly better than mode 2 in moderate E_b/N_o ratios, as the primary information is retransmitted as the secondary information, which is likely to be subject to less noise.

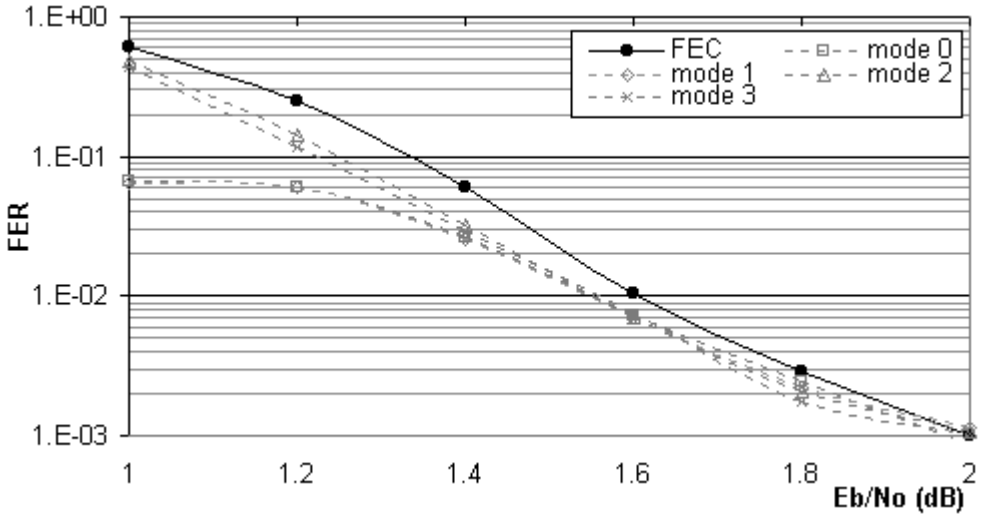


Figure 6.15 Frame error rate for N=2000 bits

For increasing E_b/N_o ratios, FER reduction of all the proposed ARQ schemes decreases. This is because the number of detected frames gets smaller (Figure 6.10 to Figure 6.12) as the majority of frame errors are corrected by the FEC scheme.

The FER of turbo codes is not as good as their BER performance. Even though increasing frame size reduces the BER, this does not ensure that the FER will reduce at the same rate. The FER results for all ARQ modes have shown that for low E_b/N_o ratios, highest coding gain is not necessarily achieved by a large frame size (i.e. N=2000 in our case). Interleaving pattern, the component codes, and the frame size, jointly determine the error correction capability of turbo codes.

6.4.2. Information Throughput

As it was discussed in section 6.3, different ARQ modes require the transmission of different amount of secondary information, which changes the overall system throughput. Overall code rates for each ARQ mode were presented in Table 6.1.

By using the ARQ probabilities calculated experimentally, system throughput has been evaluated in terms of varying code rates for all retransmission modes and frame sizes considered so far (Figure 6.16 through Figure 6.18). Maximum achievable code rate for all schemes is ½ whereas the minimum code rate for each mode is different (except for modes 1 and 2) and is displayed in all figures.

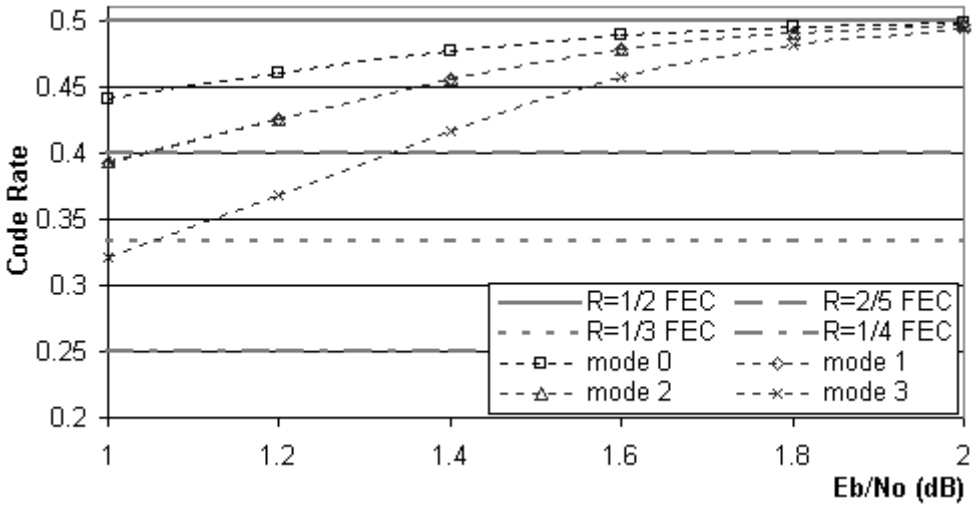


Figure 6.16 Throughput for N=512 bits

Mode 0 for all frame sizes provides the highest throughput within the 1-2 dB operation range. Mode 3 has the lowest throughput among all other schemes. Modes 1 and 2 achieve almost identical throughput, which is between modes 0 and 3. As the frame size gets larger, the maximum achievable rate is reached at lower E_b/N_o ratios since the enhanced error correction capability of the turbo code gradually eliminates the need for ARQ.

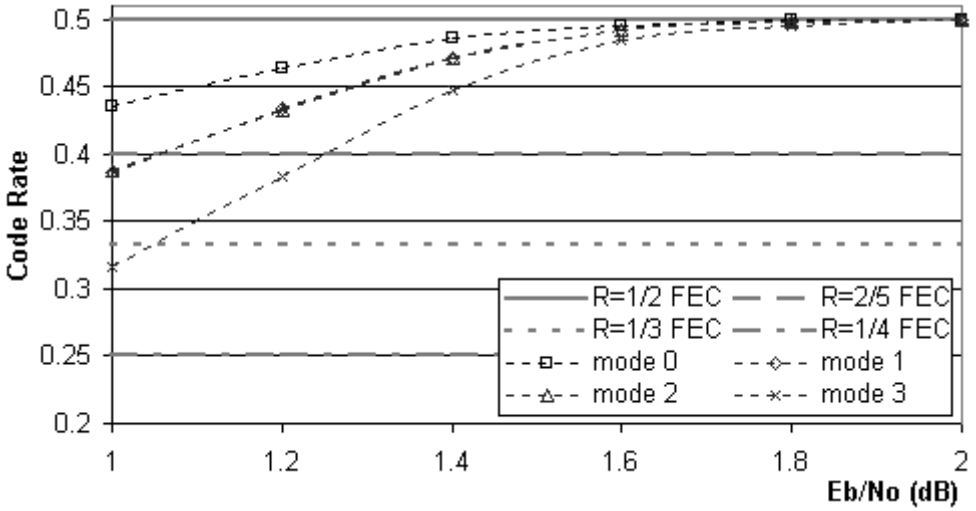


Figure 6.17 Throughput for N=1000 bits

Considering the FER and throughput results for all frame sizes, modes 0 and 1 appear to perform the best. If a choice was to be made between these two modes, the throughput requirements of the system need to be taken into account first. If the throughput difference is tolerable, mode 1 could be preferred as its FER is slightly better than that of mode 0. However, if the throughput needs to be maximized at all cost, mode 0 would be used.

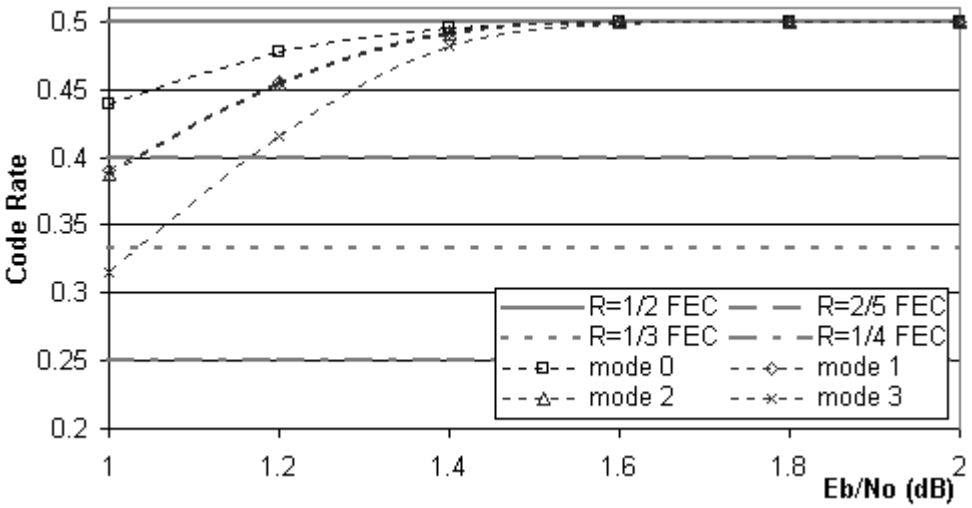


Figure 6.18 Throughput for *N*=2000 bits

6.5. Numerical Complexity

There are three main processes that have the biggest share in computational overhead introduced by the hybrid ARQ detection algorithm. They are the energy, standard deviation and average calculations, all of which involve floating point operations (multiplications and additions). In the absence of error detection, the overall decoder complexity would be mainly determined by the complexity of the component decoders, as they take up the longest processing time. Complexity of the ARQ system will be presented in comparison to the inherent decoder complexity.

The computational cost of the error detection technique introduced comes primarily from the FED block (Figure 6.8). Except the buffer, the remaining sub-blocks contribute to the computational overhead in terms of the number of multiplications and additions performed.

The squaring block transforms the buffered soft decoder outputs into energy levels by squaring them. Considering a decoded information frame with *N* information symbols, the total number of multiplications performed by this block would be *N* (Table 6.3).
Energy averages are calculated in the μ block. For a given decoded information frame with *N* symbols in it, the total number of floating-point operations are *N*-1 additions and one division.

The σ block performs *N* subtractions and *N* multiplications as well as one division and a square root operation. In order to represent the square root operation in terms of the floating-point operations, Newton's iterative method was used [Acton, 70] . This method is

iterative, and performs two divisions and one addition per iteration. From a practical point of view, in order to achieve an accuracy that is less than or equal to 10^{-7}, a maximum of 10 iterations were needed in our calculations. Therefore, the upper limit of total floating-point operations required by the square root algorithm is 10 additions and 20 divisions.

	floating-point operations per frame
squaring block	N
μ block	N
σ block	$2N+31$
request block	1
TOTAL	$4N+32$

Table 6.3 FED block calculations for a frame size N

The request block calculates the Φ detection metric and compares it with Φ_t detection threshold (see section 6.2). This block also generates the ARQ flags and performs a hard decision following a zero flag generation. As the focus of the complexity analysis is to determine the overhead introduced by the FED block, the hard decision is not taken into account, as it is a part of the classical turbo decoder structure. In addition, the flag generation and comparison operations are not floating-point operations, and hence are ruled out in our analysis. Therefore, the request block overhead can be approximated to one division per information block for calculating Φ.

For a classical turbo decoder, the component decoders introduce the highest level of operational complexity. The interleaving, deinterleaving and hard decision blocks do not perform floating-point operations and have a negligible processing overhead compared to the component decoders (typically 95% less than the component decoders). Therefore, the complexity evaluation of the component decoders represents the overall decoder complexity with acceptable accuracy.

In a component decoder, the number of floating-point operations performed depends on the number of trellis states (2^{K-1}) as well as the information frame size. Considering the state and branch probability calculations related to the max-log MAP algorithm, the total number of floating-point operations for a turbo decoder for I iterations can be expressed as in (6.4) in general form.

$$2 \cdot I \cdot (16N + 2^{K+1}) \qquad \textbf{(6.4)}$$

In the turbo decoder, which has been used in our hybrid ARQ system, K was set to 3, and a fixed number of 8 iterations were performed. When these two parameters are included in (6.4), the overall decoder complexity yields (6.5).

$$256(N+1) \qquad \textbf{(6.5)}$$

The ratio of the FED block complexity to that of the decoders indicates the processing overhead introduced by the ARQ detection algorithm described here. As the frame length, N, approaches infinity, the additional floating-point operations that are performed by the FED block approach $1/64^{th}$ of the turbo decoder's inherent calculations (6.6).

$$\lim_{N\to\infty}\frac{(4N+32)}{(256N+256)}=\frac{1}{64} \qquad \textbf{(6.6)}$$

6.6. Discussion

A novel frame error detection method that uses the soft a posteriori information has been explained. Also, a hybrid ARQ scheme that operates in four different retransmission modes has been introduced and the effectiveness of the error detection in terms of detection accuracy, information throughput and frame error reduction have been evaluated.

Results show that the frame error detection accuracy increases with the increased information frame size, which is in line with the iterative decoding principles. Since the detection metric is dependent on the soft output of the turbo decoder, the increased extrinsic information accuracy achieved by the increased information frame lengths directly improves error detection capability.

Another advantage of the new frame error detection is that it only introduces 1.56% additional computational complexity to inherent complexity of the turbo decoder.

In terms of frame error performance, mode 1 and mode 0 were observed to provide the lowest FER, whereas modes 2 and 3 provided comparable FERs, which were higher than that of modes 1 and 0.

The information throughput of mode 0 is the highest compared the other modes. Modes 1 and 2 have the same throughput, which is higher than that of mode 3.

If a choice was to be made between the 4 retransmission modes, mode 0 is the most preferable, as it provides the lowest (i.e. very close FER performance to mode 1) FER and the highest information throughput among the other modes.

The main advantage of the ARQ protocol is that it can be configured to operate in one of the four modes, which is chosen according to required throughput and frame error performance in a communication system. It is also possible to configure the turbo decoder to choose the retransmission mode depending on how severely an information frame is affected by the channel noise. However, such an application would require a wider return channel bandwidth, as additional information regarding the retransmission mode, besides the ARQ flag, would have to be sent to the transmitter.

Chapter 7

Turbo Coded Image Transmission

7. Turbo Coded Image Transmission

7.1. Introduction

In this chapter two turbo coding applications are introduced to provide reliable transmission of compressed and uncompressed digital still images under noisy channel conditions.

The first application is a serial concatenation of an RSC turbo code with the Absolute Picture Element (APEL) coding [Tanriover et al, 99a] , [Chippendale et al, 99b] , which was developed at Lancaster University. APEL is a loss-less, binary image coding technique, which provides data compression [Chippendale, 98] , [Chippendale et al, 99c] . Due to the compression introduced by the APEL coding, the coded data is susceptible to channel noise. Therefore, transmitting APEL images over noisy communication channels without an effective error control technique results in severely corrupted image data that cannot be decoded at all. In order to increase the error resilience of APEL image transmission at low signal-to-noise ratios, protection of the APEL-coded data via turbo codes has been investigated (section 7.2).

The sections of this first application is organised as follows. Section 7.2.1 explains the principles of APEL coding in detail. The incorporation of the turbo code to the APEL source coder is described in section 7.2.2 as well as the turbo coded bitmap (BMP) and the Joint Photographic Experts Group (JPEG) [Wallace, 91] images to provide benchmarks for comparison. The pixel error calculation, which quantifies the pixel disturbances of an image, is introduced in section 7.2.3, and is used for error performance evaluation. Section 7.2.4 presents the simulation results of the turbo coded APEL, JPEG and BMP image transmissions under random and the bursty channel noise conditions.

The second turbo code application is related to the transmission of the hue (H), saturation (S) and luminance (L) colour components of digital still images. Description of colour using hue, saturation and luminance has been introduced by the International Commission on Illumination (CIE) [CIE, 00] , in 1933, and relates very closely to human perception of colour. In section 7.3, an unequal error control scheme [Tanriover & Honary, 02a] , which is a modification of the two-fold turbo code introduced in Chapter 3, developed for the reliable transmission of HSL colour components over the AWGN channel is discussed. The sub-sections of section 7.3 are as follows.

The description of colour using the HSL components is explained in section 7.3.1. In addition, the visual significance of each colour component is emphasised with illustrations in the same section. The need for using an unequal error protection scheme for HSL image transmission is covered in section 7.3.2, where the modification of the two-fold turbo encoder that provides unequal error protection is also explained. In section 7.3.3, the pixel error rate is used for evaluating the performance of the HSL image transmission, where the error performance of the image transmission using a classical turbo coding scheme is compared to the transmission with the proposed unequal turbo code.

The chapter concludes with section 7.4, where the results of both turbo code applications are summarised.

7.2. Turbo-Coded Compressed Image Transmission

7.2.1. APEL Coding

A digital picture can be thought as a matrix of pixels with Y rows and X columns. Each pixel colour is represented by a finite binary number with n digits, which means that the maximum number of colours that the image can have is equal to 2^n. APEL coding has been developed for 16-colour grey-scale image, where a 4-bit word defines a pixel.

The first step in APEL encoding is to extract all the colour values from the source image, which is to be compressed. As an example let us consider the image 'tiger' shown in Figure 7.1, which is going to be APEL encoded. The word value of a pixel is denoted $H(x_i,y_i)$, where the (x_i,y_i) ordered pair represents the co-ordinates of the i^{th} pixel. The constituent bits of $H(x_i,y_i)$ is given by the sequence $\{h_{i1}, h_{i2}, h_{i3}, h_{i4}\}$ starting from the most significant bit (MSB) and moving towards the least significant bit (LSB). Note that for an image with dimensions XxY, $1 \le i \le XY$ condition is satisfied. As shown in Figure 7.1, all constituent bits in the source image 'tiger' are placed in a XY by 4 matrix.

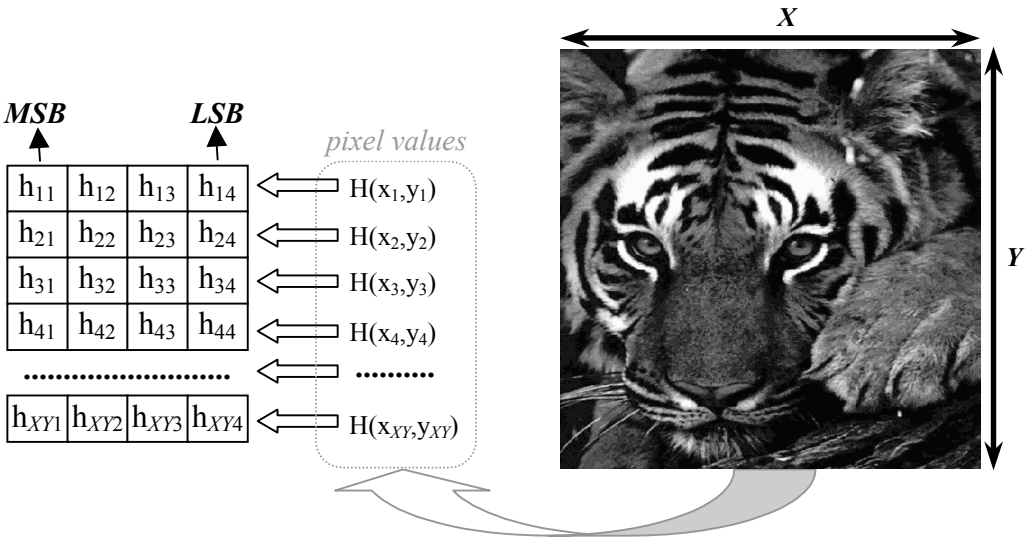


Figure 7.1 Pixel value extraction from 16-colour grey-scale image 'tiger'

The next step in compression is the bit plane coding (BPC) whereby the source image is broken down to black-and-white images. As each column in Figure 7.1 is considered to contain data for one image, 4 such images are constructed (Figure 7.2). In this case, a column bit is treated as a colour value, which is either '1' for black or '0' for white pixels. By using BPC, every 2^n-colour source image can be broken down to n black-and-white images, each of which has the same size with the source.

Following the BPC stage, each bit plane is APEL encoded. APEL identifies black pixel groups, which form squares of various dimensions, called *picture elements* (pels). Each pel is defined by the length of one side in units of pixels as well as a unique address indicating its location on the image. Starting with the largest pels, each bit plane is scanned and all available pels are identified. Consider, as an example, pel extraction from a part of the third bit plane from 'tiger' (Figure 7.3). In the chosen image section, the largest square has 4 pixels per side and its bottom-left pixel is located at (3,1) co-ordinate. Therefore, this pel can be identified with $S(4,3,1)$ tag, where 'S' indicates a square pel, and the numbers are the side length of the pel and its co-ordinate on the bit plane, respectively. The next largest square on the same bit plane is encoded in a similar way with the $S(2,1,3)$ tag.

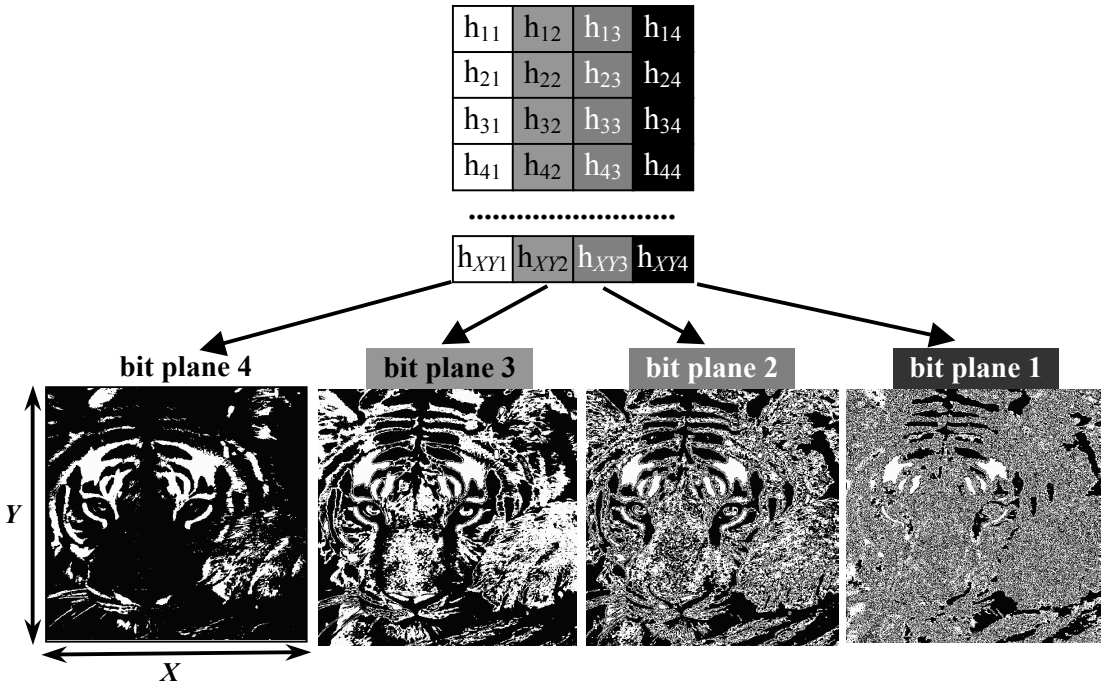


Figure 7.2 Bit plane coding of the image 'tiger'

Once all the pels are encoded, the bit plane is left with residual pixel trails termed as the *runlengths*. As a part of APEL's loss-less compression feature, the runlengths are also encoded.

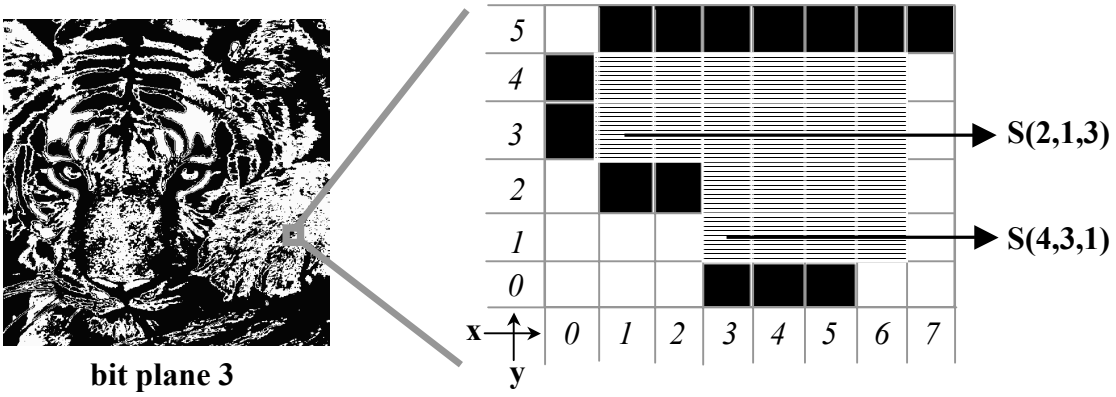


Figure 7.3 Square search in APEL coding

Figure 7.4 illustrates the runlength coding of the residual sections after the removal of pel from the third bit plane in Figure 7.3. Starting with the horizontal runlengths, the longest ones are identified and tagged as in the square pels. After identifying and tagging the horizontal runlengths (*H*), vertical runlengths (*V*) are also coded in a similar way. In our example, the longest horizontal runlength is 7 pixels long and its left most co-ordinate is (1,5), and is therefore tagged as *H*(7,1,5). The last two horizontal runlengths of the bit plane are *H*(3,3,0) and *H*(2,1,2). The only vertical runlength in our example is two pixels long and is tagged as *V*(2,0,3).

After encoding the pels and the runlengths in all four bit planes, as described above, the information is organised as packets prior to transmission. Each data packet consists of two parts; the control symbol and the pel addresses (Figure 7.5).

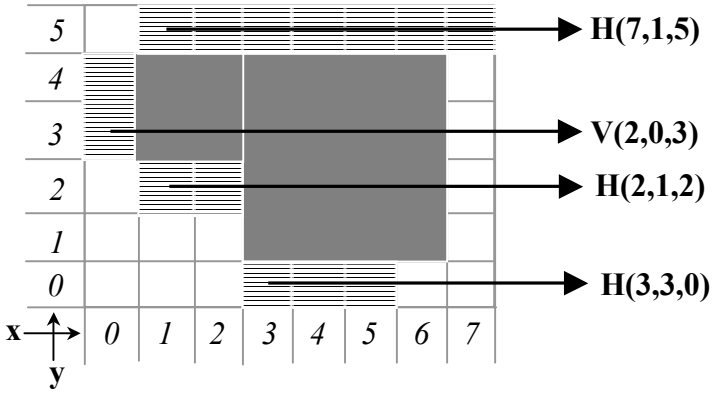


Figure 7.4 Horizontal and vertical runlength search in APEL coding

The control symbol stores information related to image and pel dimensions, synchronisation, and bit plane, to which the pel addresses belong.

The pel addresses are ordered in the same way as they are encoded, i.e. address of squares are packetised first, followed by the horizontal and vertical runlengths. Note that large sizes also have the priority during packetising. Therefore, $p(x_1,y_1)$ would represent the address of the largest square pel in one of the bit planes.

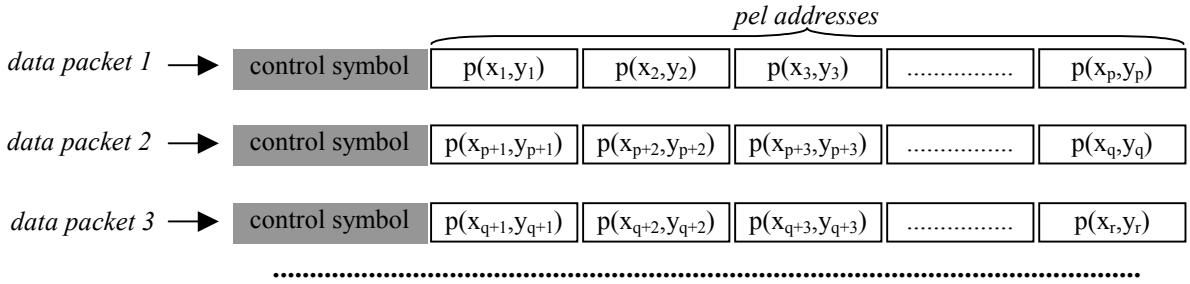


Figure 7.5 APEL data packet structure

At the receiving end, as the data packets arrive, the APEL decoder progressively constructs the source image by interpreting the size and the location of each pel. Since the large pels are transmitted first, the decoder is able to generate a low-resolution image even though the entire image information is yet not received.

An example of the progressive decoding is presented in Figure 7.6 for the image 'tiger' that has been APEL encoded. After receiving only 5% of the source image, it is possible to visualise a significant amount of image detail already. By decoding 20% of the APEL encoded image, almost 90% of the entire image detail is obtained. In fact, the remaining

80% that is not yet received have very little to contribute to the already acquired image information by the human eye. As can be seen from Figure 7.6, image resolution is similar for 100%, 70% and 20% data reception.

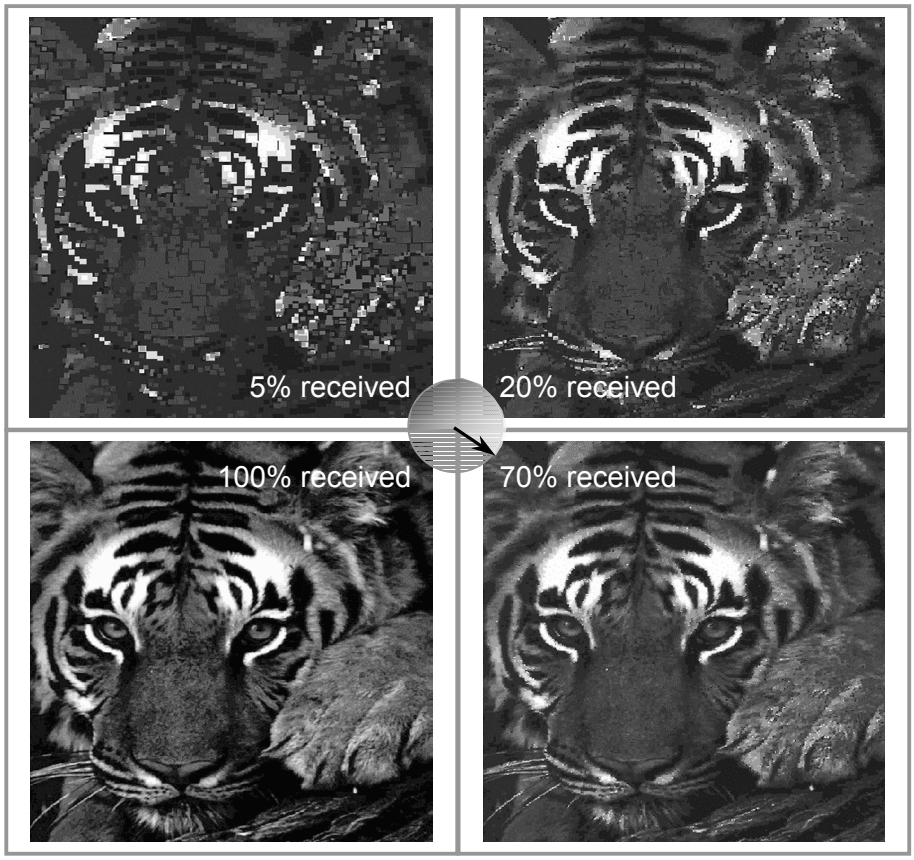


Figure 7.6 Progressive decoding of APEL encoded source image 'tiger'

The progressive decoding feature of APEL is particularly useful in a situation where the data transmission is unexpectedly interrupted due to a failure in the communication link. For the sake of argument, let us assume that 30% of an APEL image is received immediately before a transmitter malfunction. In this case, the receiver will have received enough information without having to wait for the remaining 70% of the data. More importantly, after the transmitter is ready to send information again, the transmission can continue from the point where it was interrupted. Therefore, the received 30% image data is not retransmitted, and hence the available bandwidth is used efficiently.

7.2.2. Turbo Coded APEL Transmission

In order to assess the performance of the APEL transmission with forward error correction, the simulator in Figure 7.7 was used. First, a source image is APEL encoded, as explained in section 7.2.1. Then the APEL data packets are fed to the g(7,5) RSC turbo encoder. The BPSK modulated codewords are transmitted over an AWGN channel, and are demodulated

at the receiver. Prior to APEL decoding, the turbo decoder attempts to correct the majority of bit errors. Finally, the received pels are ordered and placed onto their correct bit planes, after which the decoded image is compared to the source image for determining the pixel error rate (PER).

To demonstrate the benefits gained and also to provide benchmarks for comparison, in addition to incorporating turbo coding into an APEL system, two additional image formats have also been concatenated with the same turbo code, as illustrated in Figure 7.7. The first one is a widely recognised compression scheme, i.e. JPEG. The JPEG images have high quality, despite their heavy compression levels (up to 99 to 1). The second image file format used is the BMP, which is uncompressed, and the pixels are uncorrelated.

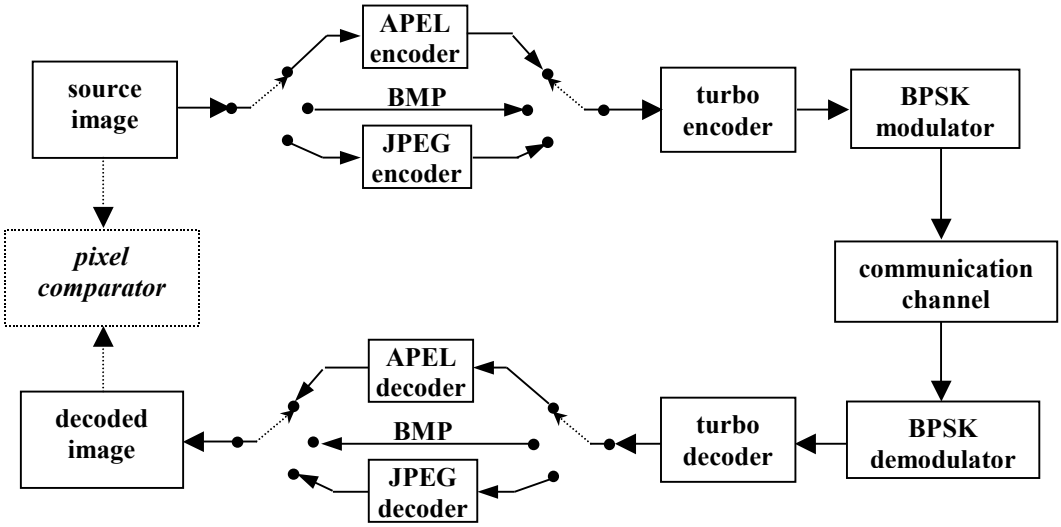


Figure 7.7 Turbo coded image transmission simulator

One important point that needs to be considered while comparing the turbo coded APEL and the JPEG schemes is to make sure that the source images encoded by both schemes are compressed to a similar size. Figure 7.8 shows the compression levels of the three image formats used. As the BMP format offers no compression, it has a compression ratio of 1. The JPEG and the APEL images, on the other hand, are reduced to one third of the BMP file, and hence have a compression ratio of 1/3.

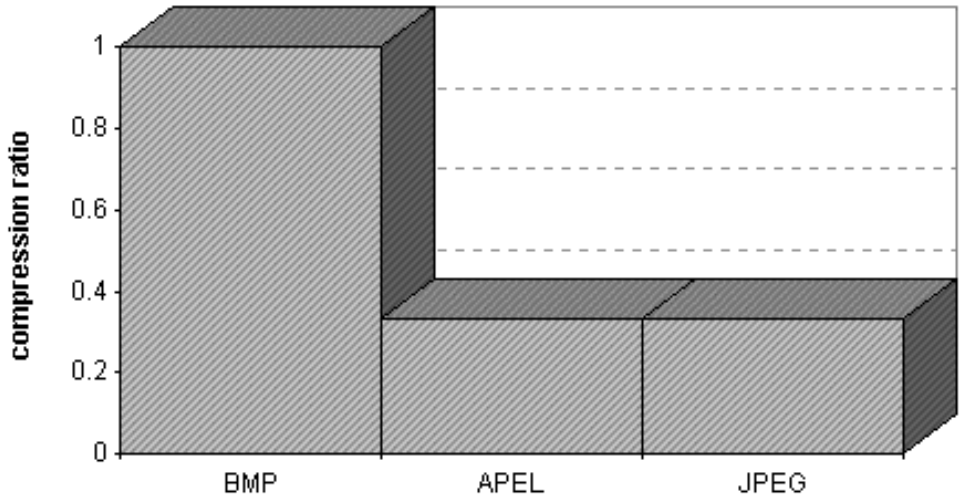
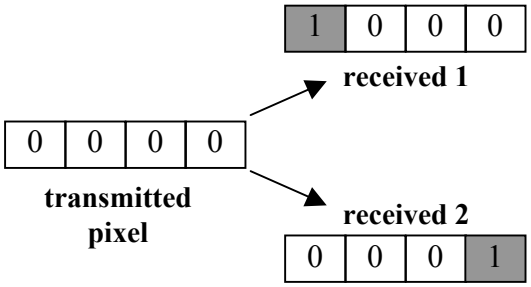


Figure 7.8 Compression levels of the image formats

7.2.3. Visual Impact of Bit and Pixel Errors

In error control coding, the error performance is measured using BER, which is the ratio of the number of encountered bit errors to the number of decoded information bits. However, for images, besides the number of bit errors, error locations in pixels also need to be reflected to the error performance.

To clarify the significance of the number of bit errors and their location in a pixel, assume that for a 16-color image, a transmitted pixel value {0000} is affected by noise in two different ways (Figure 7.9). In the first received version, noise has inverted the MSB, whereas in the second one the LSB is received in error. If we choose to evaluate the error performance in terms of BER, as both received pixels have one bit in error out of 4, the BER will be 0.25. However, to the human eye those two pixels will convey two different degrees of error. In terms of colours, the first received pixel is 7 colours away from the transmitted pixel, whereas for the second received pixel this difference is only 1.



	bit errors	color errors
received 1	1	7
received 2	1	1

Figure 7.9 Colour and bit error examples for a pixel

Let us now take the above example a step further and visualize how such pixel errors actually look on a 16-color grey-scale image. To do this, the source image 'tiger' has been modified according to the example in Figure 7.9. In the first and the second received images, the MSB and the LSB of all pixels are inverted, respectively (Figure 7.10). Note that for both received images, the BER is 0.25, which indicates that they have identical number of bit errors. Visually, on the other hand, one can easily see that the picture quality of the second received image is closer to the source image. This simple example illustrates the need for an alternative ratio for evaluating the error performance for image transmission, namely the PER.

In order to quantify the visual disturbance (Δ) introduced by the channel noise, the colour difference between the received and the transmitted pixels needs to be calculated. Given that the number of colours in a digital image is n, the maximum colour error can be $n-1$. If, for a pixel i, the transmitted and the received pixel values are denoted as, t_i and r_i, respectively, the visual disturbance for i can be calculated as in (7.1).

$$\Delta_i = \frac{|r_i - t_i|}{n-1} \qquad \textbf{(7.1)}$$

For a transmitted source image with XxY pixels, the PER can be calculated as in (7.2). Note that the PER can be seen as the average visual corruption on a digital image, and it is a close numerical representation of the error perception of the human eye.

$$PER = \frac{1}{XY}\sum_{i=1}^{XY}\Delta_i \qquad \textbf{(7.2)}$$

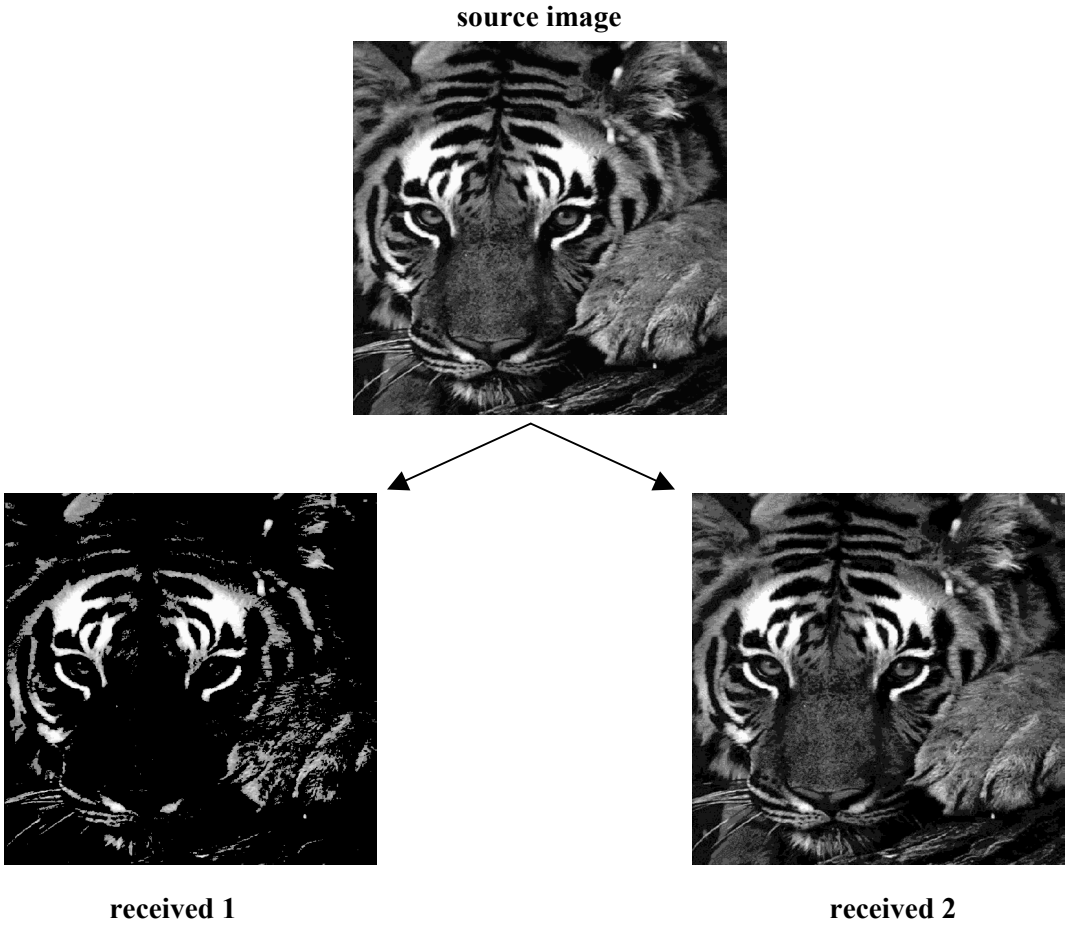


Figure 7.10 Visual difference between received images with identical BERs

7.2.4. Error Performance

The error performance of the turbo coded image transmission schemes was evaluated over the AWGN as well as a bursty channel, which generates random bursts of errors up to 200 bits per information frame. In a transmitted codeword the location and the duration of those bursts are random. For all the simulations, the code rate was fixed at ½ and the interleaving degree was 8000. In addition, the number of decoder iterations was fixed at 16.

For the AWGN simulations the BMP, APEL, and the JPEG versions of the source image 'cat' were transmitted (Figure 7.11). Figure 7.12 shows the PER performance for the three turbo-coded image transmission schemes. Note that each error curve represents the average PER data of 100 image transmissions. Multiple transmissions were observed to increase the reliability of the error performance analysis.

Figure 7.11 Source image 'cat'

As Figure 7.12 shows, the performance of the turbo-JPEG scheme is very poor in the 1.0-1.4 dB range. This is due to the inherent fragility of the JPEG structure and its inability to correct or restrict the propagation of any errors. Turbo-BMP scheme, on the other hand, is good throughout the range. This results from the complete independence of all pixels from one another. Hence, when the turbo decoder cannot fix certain bit errors, only pixels with the corrupted bits are affected. Finally, the turbo-APEL performance is comparable to the turbo-BMP scheme, even though it is compressed. In fact, as the channel conditions improve after 1.18 dB, the turbo-APEL performs better than the turbo-BMP concatenation. This is due to the post-processing techniques used by the APEL decoder, which effectively corrects pel errors rather than recovering individual pixels.

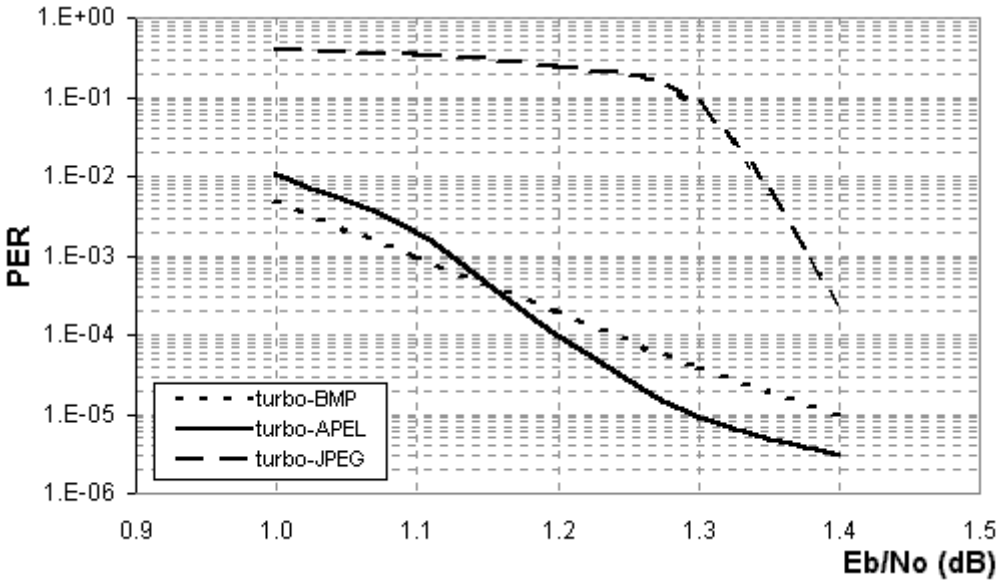


Figure 7.12 Pixel error performance over the Gaussian channel

To observe the visual impact of data errors, samples of BMP, APEL and JPEG images transmitted at 1.0, 1.2 and 1.4 dB have been decoded and are included in Figure 7.13 through Figure 7.15.

The bit errors in the BMP images affect the pixels individually whereas due to the bit plane coding, errors in the APEL images appear as squares and run-lengths of various sizes. Corruption in the JPEG images is the most severe due to its fragile header structure and high inter-pixel correlation.

The visual quality of the APEL images clearly improves as the noise level decreases. More importantly, at 1.4 dB (Figure 7.15) the decoded APEL image almost looks the same as the source image in Figure 7.11. In addition, the BMP and APEL image quality converges as the channel conditions improve.

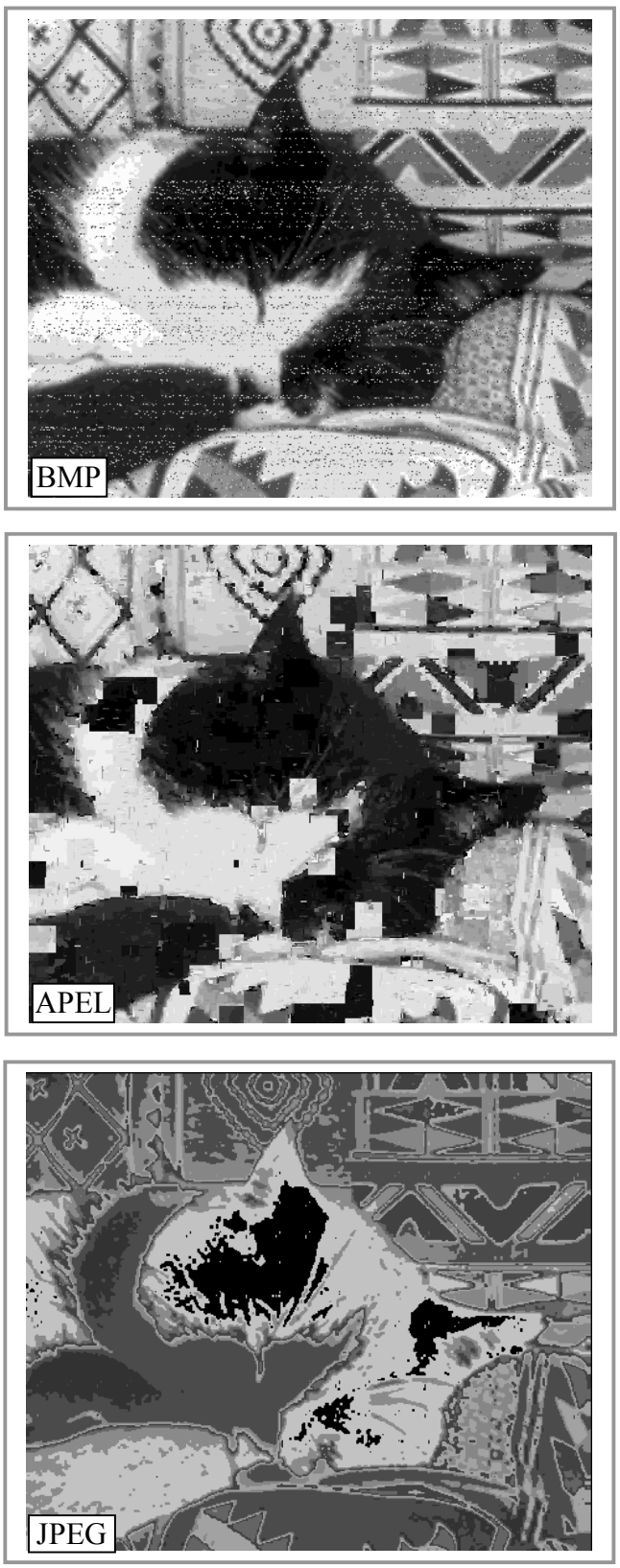


Figure 7.13 Turbo coded transmission over Gaussian channel at E_b/N_o=1.0 dB

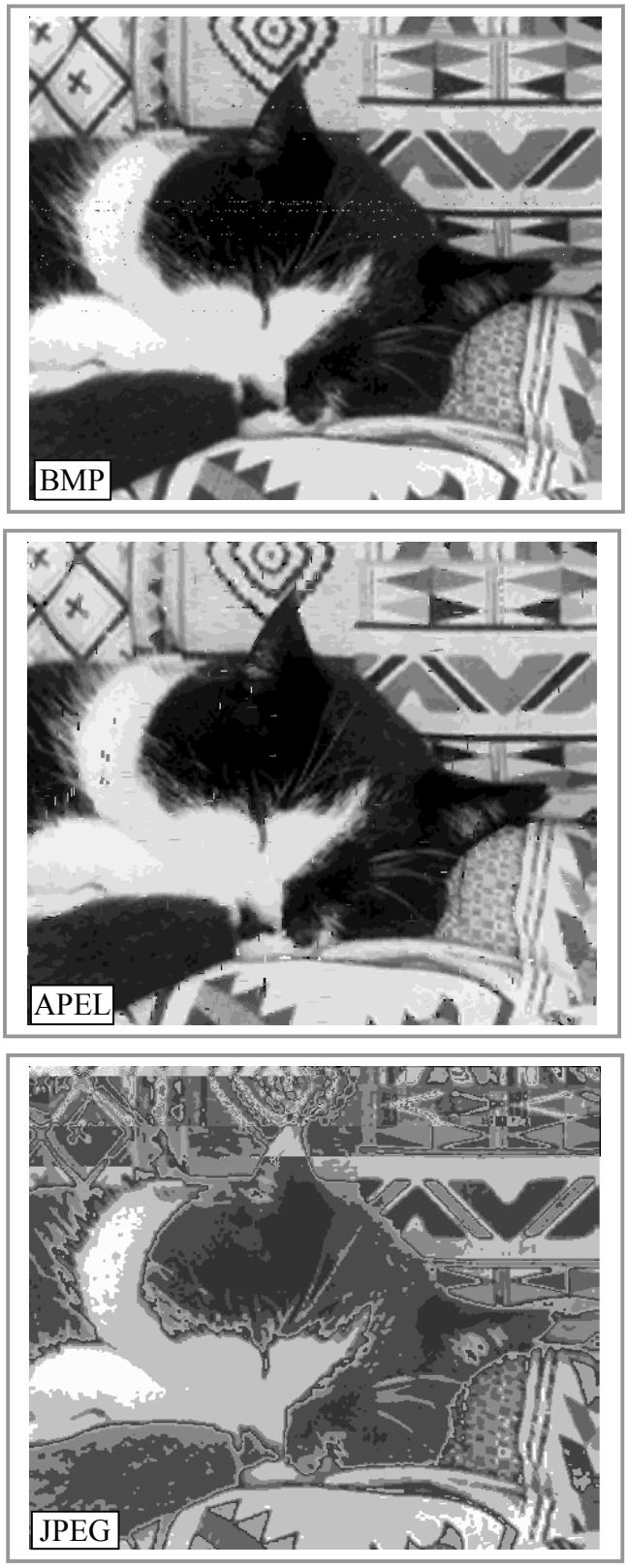


Figure 7.14 Turbo coded transmission over Gaussian channel at E_b/N_o=1.2 dB

Figure 7.15 Turbo coded transmission over Gaussian channel at E_b/N_o=1.4 dB

In addition to the Gaussian noise, random bursts were also introduced to the channel. Using a new source image, namely 'Enzo' (Figure 7.16) the turbo coded BMP, APEL, and JPEG image transmissions were repeated. The PER performance for the bursty channel (Figure 7.17) was also averaged from 100 different image transmissions. As observed during the simulations, despite the turbo coding stage, the JPEG images were severely

corrupted by the bursts of bit errors introduced. Since the received JPEG images could not be decoded for the pixel error analysis, they had to be discarded.

Figure 7.16 Source image 'Enzo'

As Figure 7.17 shows, turbo-BMP scheme performs better than the turbo-APEL scheme for signal-to-noise ratios less than 4.0 dB. Beyond 4.0 dB, the turbo-BMP scheme performance improves slowly, which is due to the turbo decoder's limited capability of combating burst errors. The turbo-APEL scheme also follows a similar trend; after about 4.0 dB the slope of the PER curve starts to decrease. However, in comparison to the turbo-BMP case, the PER of the turbo-APEL scheme reduces faster with the improving channel conditions. The reason for this is the APEL decoder's capability of correcting pels by using the pixel geometries embedded within the APEL encoded image. In other words, the turbo decoder's bit-based error correction power is supplemented by the APEL decoder's pel recovery techniques, which makes the image transmission scheme more robust to burst errors.

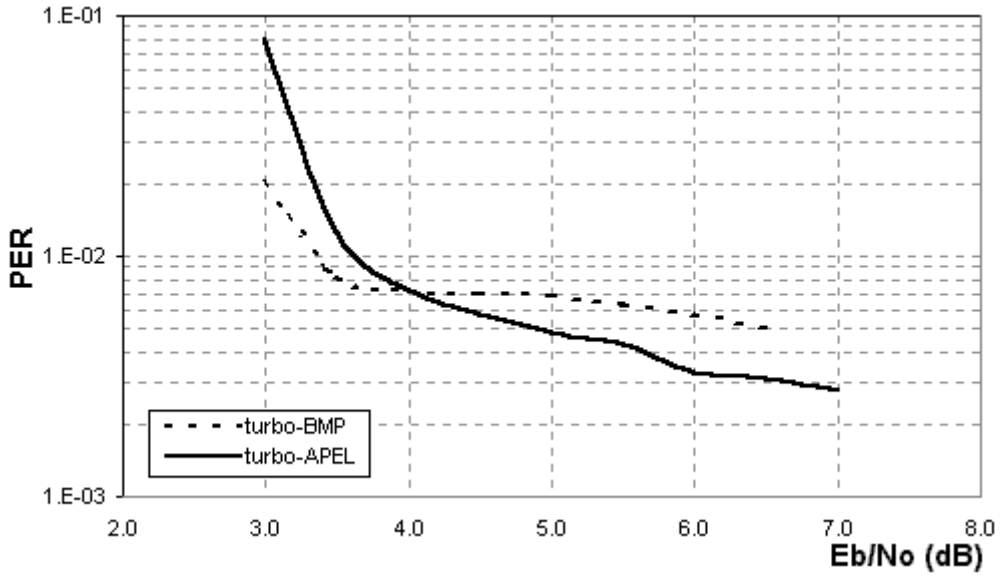


Figure 7.17 Pixel error performance over the bursty Gaussian channel

The APEL and BMP images, transmitted over the bursty Gaussian channel at 3.0, 5.0 and 7.0 dB are decoded and are shown in Figure 7.18, Figure 7.19, and Figure 7.20, respectively.

As illustrated in the BMP images, the channel errors are introduced both randomly and in bursts. In all the BMP results, the majority of the random errors are eliminated, especially when the E_b/N_o moves from 3.0 dB towards 7.0 dB. However, the uncorrected burst errors appear as pixel trails distributed randomly over the BMP images. Since the pixels in a BMP image are uncorrelated, such erroneous pixel trails cannot be fixed and cause the aforementioned error floor in the turbo-BMP PER curve (Figure 7.17).

In the case of APEL decoded images, the distribution of errors does not follow any noticeable patterns at all, and appear to be randomly distributed over the images. This is due to the variable transmission order of pels in different bit planes, as explained in section 7.2.1. The progress of the APEL picture quality under improving channel conditions can clearly be seen in Figure 7.18, Figure 7.19, and Figure 7.20. By using the valid image information in certain bit-planes, the APEL decoder can recover the damaged pels in other planes, which can be a powerful tool for eliminating burst errors.

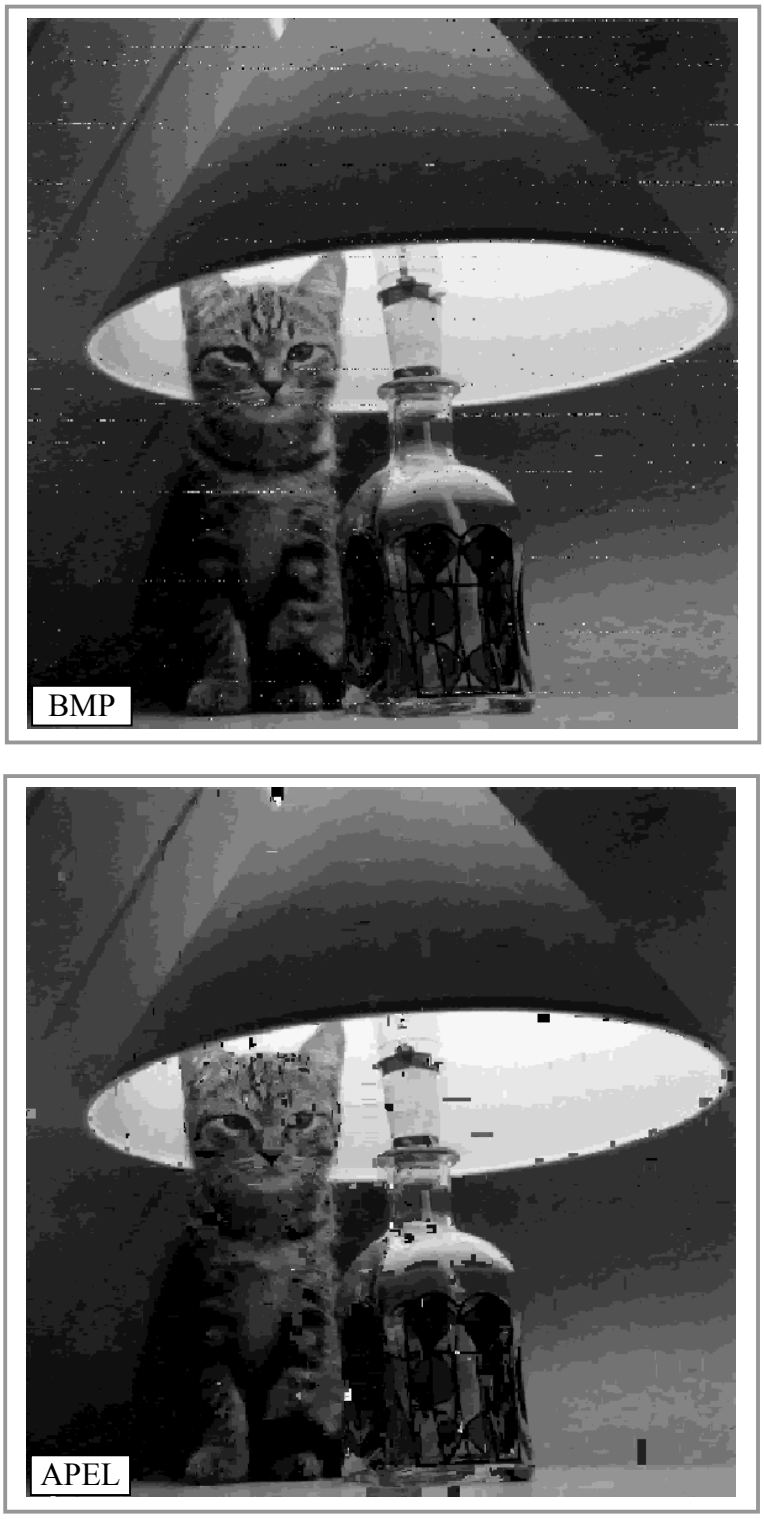


Figure 7.18 Turbo coded transmission over bursty Gaussian channel at E_b/N_o=3.0 dB

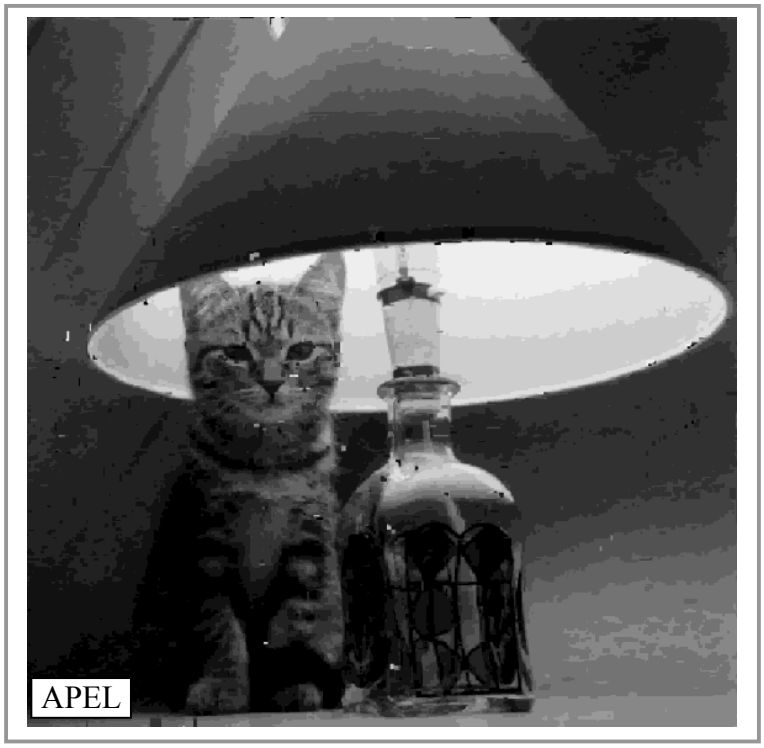


Figure 7.19 Turbo coded transmission over bursty Gaussian channel at E_b/N_o=5.0 dB

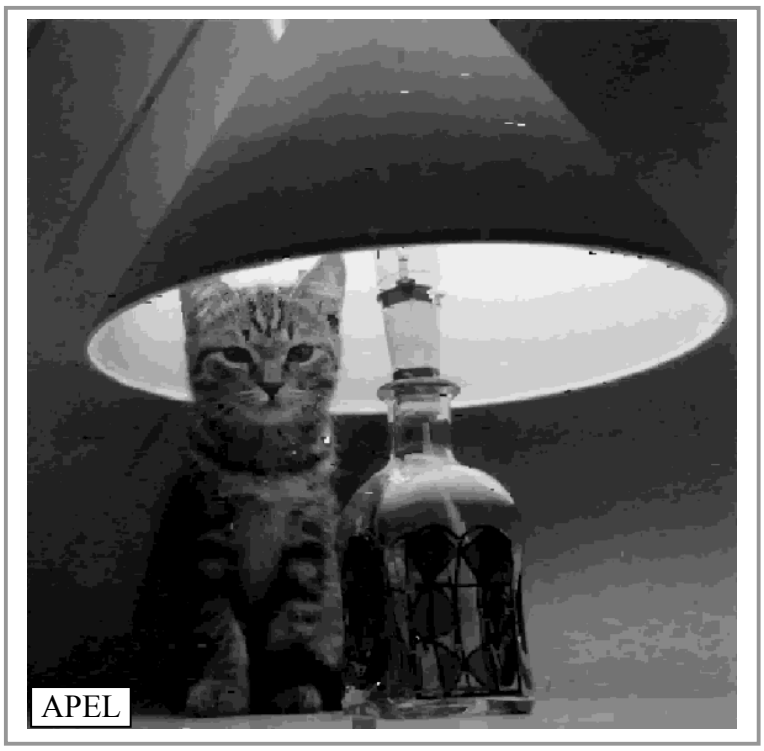


Figure 7.20 Turbo coded transmission over bursty Gaussian channel at E_b/N_o=7.0 dB

When evaluating the image performance, the most appropriate tool to use is the human eye; after all the final recipient of the images is a human. Keeping this point in mind, a psychometric study based on the subjective rating of a group of 21 individuals was conducted.

Each person was shown turbo coded APEL and BMP image files transmitted over the bursty Gaussian channel, and was asked to rate each decoded picture in a scale from 1 to 10, where 10 indicates the best image quality. This procedure was repeated for different channel conditions between 3.0 and 7.0 dB, and the gathered data was averaged and plotted against the E_b/N_o as the average perceptual quality (Figure 7.21).

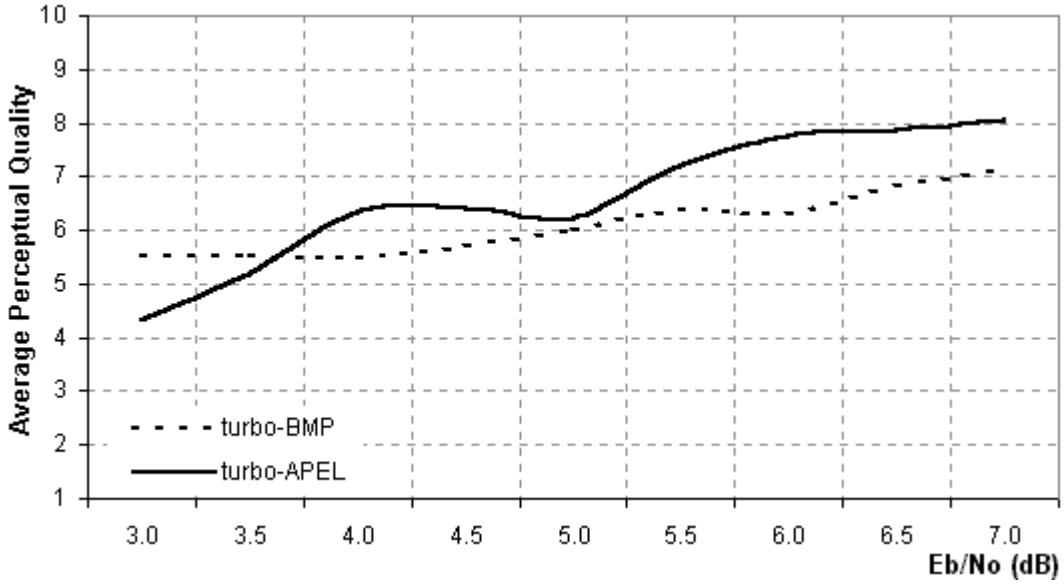


Figure 7.21 Psychometric performance over the bursty Gaussian channel

The psychometric results were observed to agree with the PER curves in Figure 7.17. Small perturbations in the perceptual quality curves of the turbo-APEL scheme (between 4.5 and 5.5 dB) were found to be due to location of the damaged areas in the images used during the experiments. It was found that when the errors are concentrated on the important image detail, i.e. the vicinity of the face in image 'Enzo', they tend to disturb the human eye more [Tanriover et al, 99a] . This can be seen in Figure 7.22 where at 5.0 dB several clustered APEL errors obscure an area of Enzo's face.

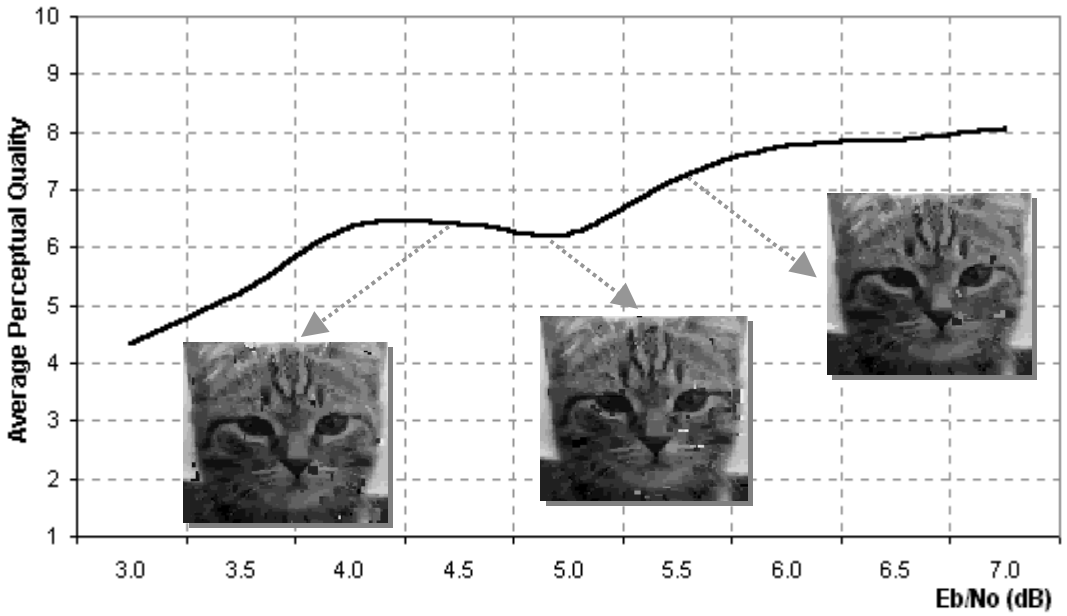


Figure 7.22 Pixel error location-perceptual quality relationship in an APEL image

7.3. Unequal Error Protection with Two-fold Turbo Codes

7.3.1. HSL Description of Colour

Description of colour using hue, saturation and luminance has been introduced by the International Commission on Illumination (CIE), in 1933, and relates very closely to human perception of colour.

HSL colour coding can be visualised as a cylinder, which has been constructed by stacking infinite number of equally sized discs (Figure 7.23).

The height of the cylinder corresponds to the L component, and varies from 0 to 100%. Therefore, the uppermost disc on the stack has 100% luminance, and is white, whereas the disc at the base of the cylinder has 0% luminance, and is black. In other words, the light levels on a digital image are stored in the luminance component.

The H and the S components are described according to a point located on the surface of a disc. The H component is defined by the angle that is swept as one moves around the perimeter of the disc relative to 0^o. Conventionally, red is located at 0^o, and green, cyan and blue are located at 90, 180 and 270^o, respectively.

The last HSL component, namely 'S', is determined by the length of the radius. For instance, if hue is 0^o, and the radius is at its maximum length, full red saturation is selected.

If we move closer to the centre of the disc without varying the hue, saturation of red decreases and it gradually transforms into grey. In fact, the centre of the disc, where the saturation level is minimum, is grey.

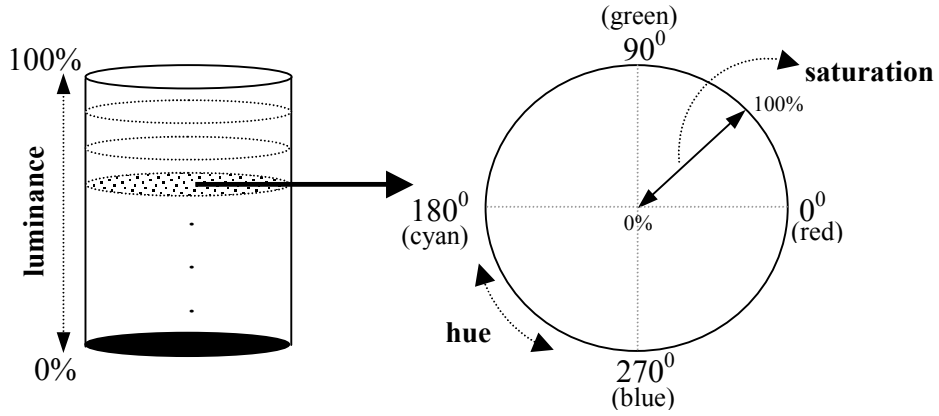


Figure 7.23 Hue, saturation and luminance components of colours

Having described the HSL colour transformation briefly, let us now consider a sample image in order to visualize the three colour planes. Figure 7.24 shows the source image 'pathfinder' and its individual colour components. Each pixel in the source image is represented by three bytes (one byte per HSL component). When all the bytes for a component are mapped as a pixel, a grey-scale image of that component can be constructed. The individual HSL images are particularly useful in a sense that they visualize how much information pertaining to image detail each component contains. Although the hue and the saturation components in Figure 7.24 jointly provide useful information related to the source image, the level of detail is limited when the two components are considered individually. This is because the hue and the saturation of a source image do not change too fast, and hence the H and S images have visually static areas.

The human eye is known to be more sensitive to light intensity changes than to colour changes, which explains the importance of the luminance component. Comparison of the luminance plane and the source, 'pathfinder', in Figure 7.24 unveils the strong correlation between the two. In fact, by only looking at the luminance component, one can easily perceive the significant visual detail of the source image. Recall that the luminance component of an image is associated with the light level of a pixel.

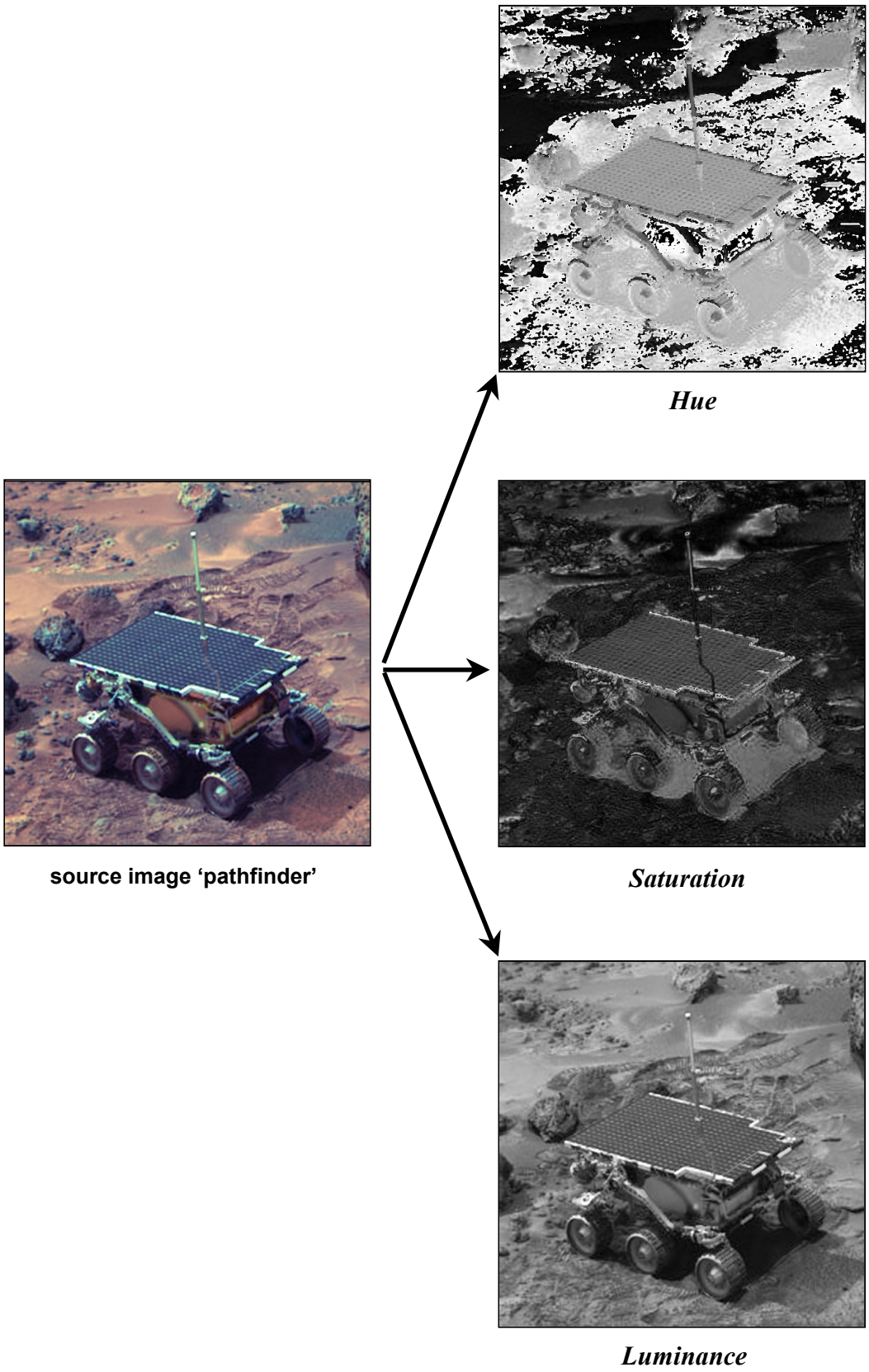


Figure 7.24 HSL components of the source image 'pathfinder'

Therefore, the grey-scale version of a source image has the same information as its luminance component, which explains the close link between the source image and the luminance plane.

Prior to transmitting the HSL components of a digital image over a noisy channel, each colour component needs to be channel coded to withstand the effects of channel noise. Moreover, extra protection should be introduced for the luminance component in particular, as the human eye is highly sensitive to that colour element, and therefore the pixel errors on the luminance plane would have a more corruptive effect than those in the H and S planes.

In order to pinpoint the sensitivity of the human eye to each colour component, Figure 7.25 is included, where each colour component of the image 'pathfinder' has been corrupted by noise. In each image the magnitude of the 20% of all pixels have been randomly modified, and hence noise has been added. Despite the equal level of degradation on each plane, the pixel disturbance on the luminance plane is more noticeable due to the high sensitivity of our eyes to changes in light levels. The errors on the hue and the saturation planes, on the other hand, are less pronounced as the human eye is less sensitive to those components.

In the next section, the modification of two-fold turbo coding [Tanriover et al, 01b] will be discussed to provide the necessary unequal error protection for reliable transmission of HSL colour information.

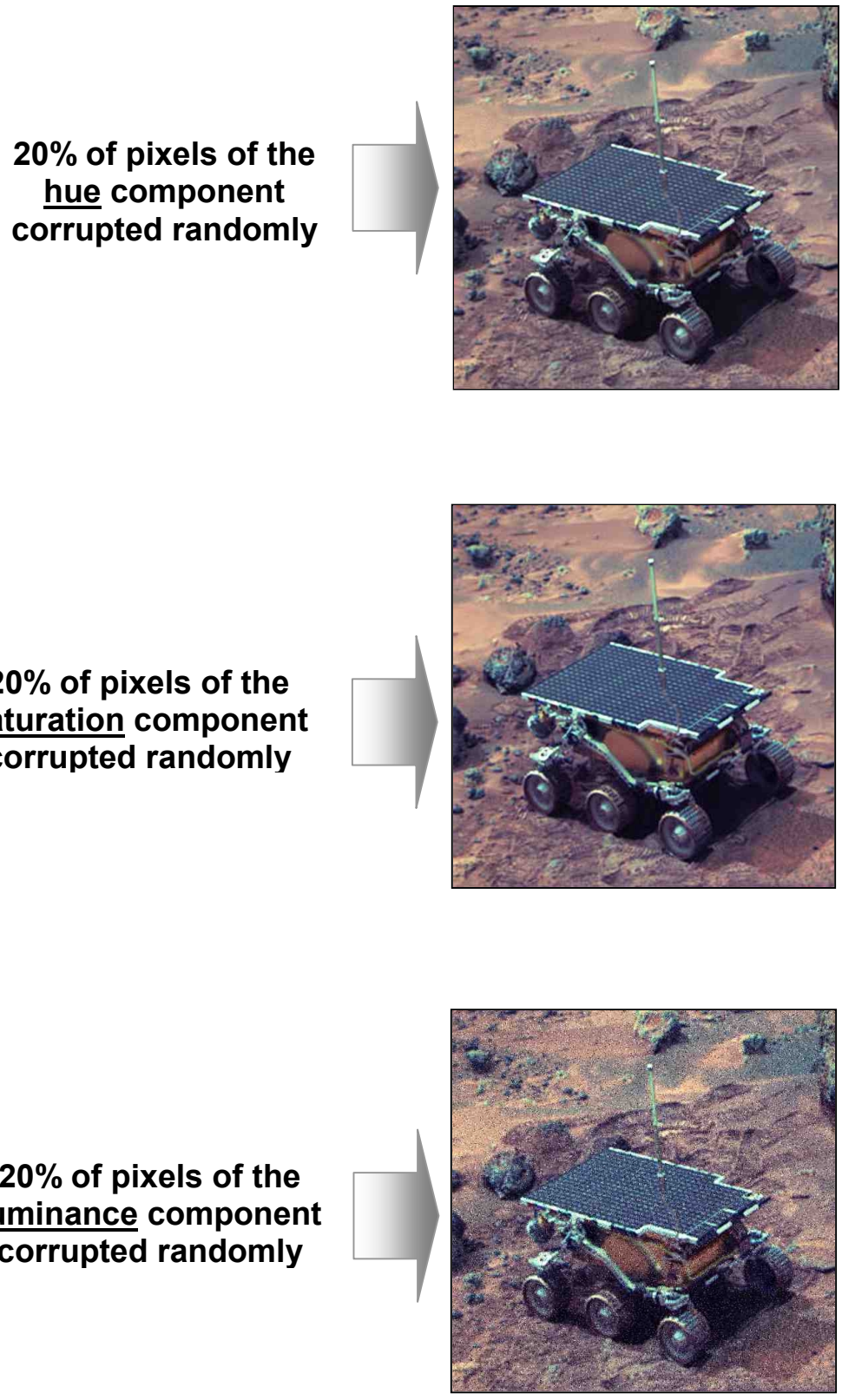


Figure 7.25 Error sensitivity of the individual HSL components

7.3.2. Modified Two-fold Turbo Coding and Unequal Error Protection

The two-fold turbo codes were introduced in section 3.4 as a member of the multifold turbo codes with the least decoding complexity. In this section, the modification of a two-fold turbo encoder to provide unequal error protection [Tanriover & Honary, 02a] , which is suitable for HSL image transmission, will be explained.

The two-fold turbo encoder and its modified version can be designed as in Figure 7.26. An information frame $I\{123\}$, consists of three segments, each containing L_S bits. The segments, namely 1, 2, and 3, are permuted in groups of two to generate three different sub-frames, namely $I\{A\}$, $I\{B\}$, and $I\{C\}$. The length of each sub-frame determines the interleaving degree, i.e. N_I, which is equal to $2 \cdot L_S$. Two interleavers (Π_1 and Π_2) are used to separate the sub-frames, prior to encoding. Three parity codewords, each with length N_I, are generated ($P\{A\}$, $P\{B\}$, $P\{C\}$) after encoding. In two-fold turbo encoding, A, B and C correspond to the segment combinations 23, 12 and 13, respectively. As the code is systematic, $I\{123\}$ also appears as a part of the output codeword. Note that with the two-fold coding scheme, each segment is encoded twice, and is therefore protected equally.

However, the two-fold code can be modified to offer unequal error protection when the segment combination A is changed to '123' with B and C staying the same. As a result of this variation, the first segment is encoded three times, whereas the second and the third segments have equal error protection and are encoded twice.

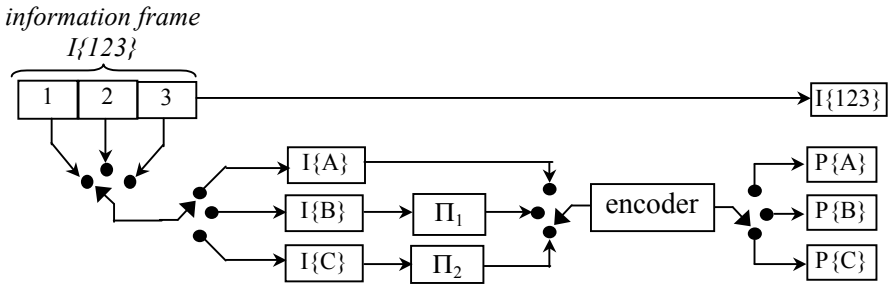


Figure 7.26 Generic two-fold turbo encoder

As far as the transmission of HSL components is concerned, the first segment of the information frame in the modified two-fold coding scheme offers additional error protection for error coding the luminance layer. Segments two and three, on the other hand can be dedicated to the transmission of the hue and the saturation information.

Having explained the encoder modifications for unequal error protection with two-fold turbo codes, due to their straightforward nature, the necessary decoder modifications are not discussed here. The multifold turbo decoding covered in section 3.3.3 provides the necessary information to build variations of any *M*-fold decoder.

7.3.3. Error Performance

The PER (7.2) performance of HSL transformed 'pathfinder' image has been evaluated using a classical turbo code (*CL*) and its modified two-fold equivalent (*MTF*). The unpunctured code rate for the *CL* is equal to 1/3, whereas for the *MTF* it is 3/10. A moderate information frame size of 4608 bits was chosen for both schemes. In order to minimize the decoding latency, the 4-state g(7,5) RSC code was decoded in parallel using the max-log MAP algorithm, and the maximum number of iterations was fixed at 16.

Figure 7.27 presents the comparative PER performance of the *CL* and the *MTF* for 2, 4, 8 and 16 iterations. Note that the PER has been calculated after combining the decoded H, S and L components and comparing the reconstructed image with the transmitted source image. It can be seen that at PER=10^{-5}, 4 MTF iterations (4i-*MTF*) provides about 0.7 dB gain over the *CL* scheme (4i-*CL*). More importantly, at E_b/N_o ratios higher than 0.75 dB, the pixel corruption on the decoded images with 4 *MTF* iterations is far less than that of the 8 and 16 CL iterations.

In order to provide a visual comparison between the *CL* and the *MTF* turbo coded HSL images, a set of decoded 'pathfinder' images with 2,4,8 and 16 iterations have been presented in Figure 7.28 through Figure 7.31. The superior picture quality of the *MTF* images compared to the *CL* images can clearly be seen - especially for 2 and 4 iterations. When the PER decreases below 10^{-4}, the pixel errors become more difficult to notice, which can be seen in the *MTF* and *CL* images decoded with 8 and 16 iterations (Figure 7.30 and Figure 7.31).

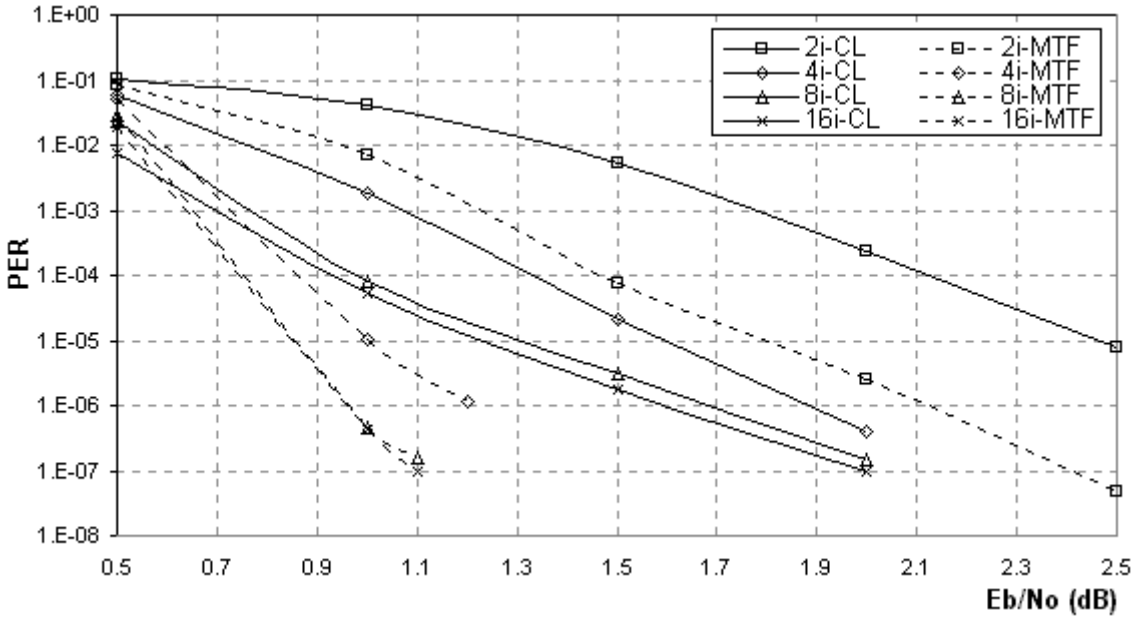
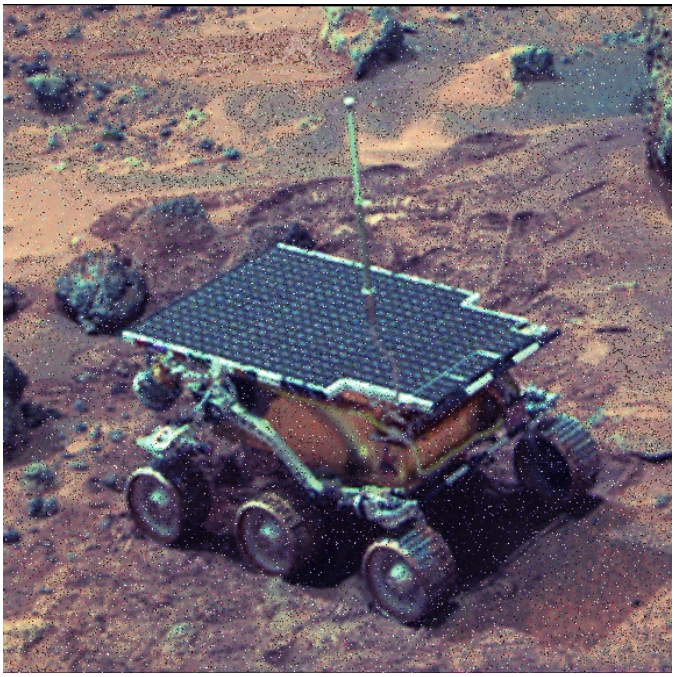
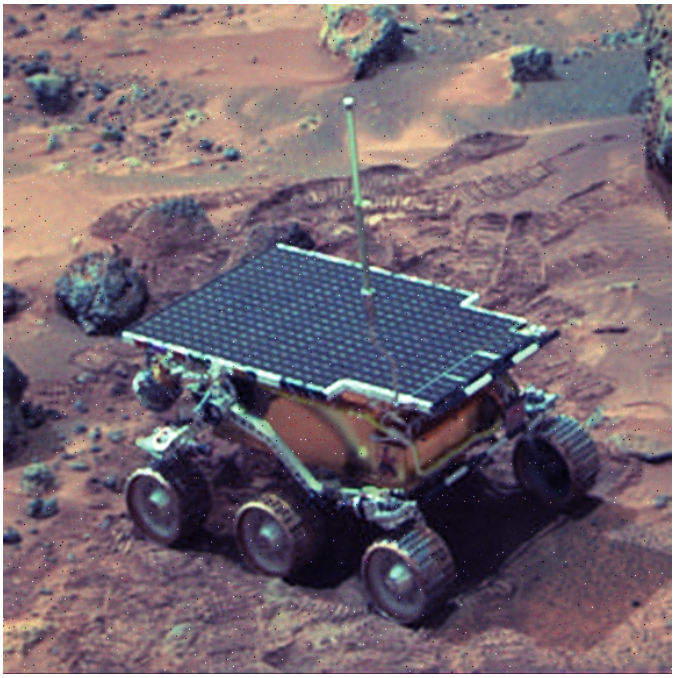


Figure 7.27 Pixel error performance

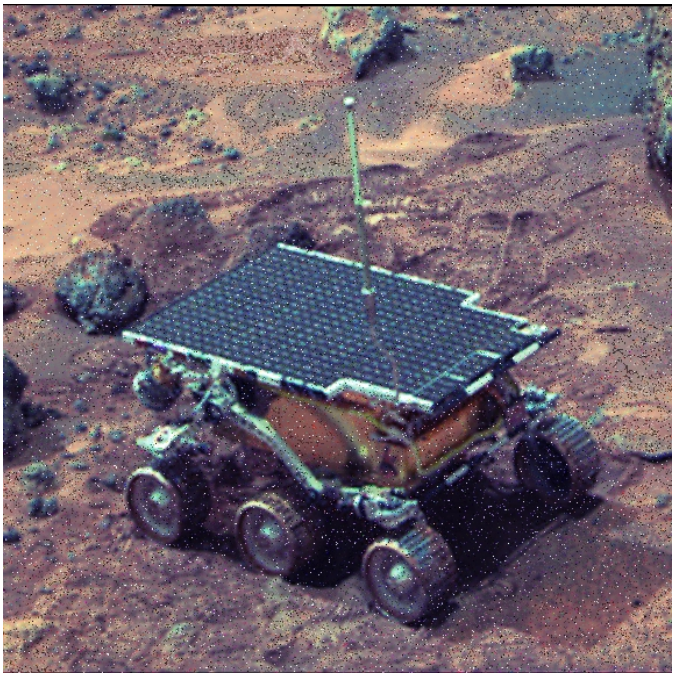


(A) Classical turbo scheme

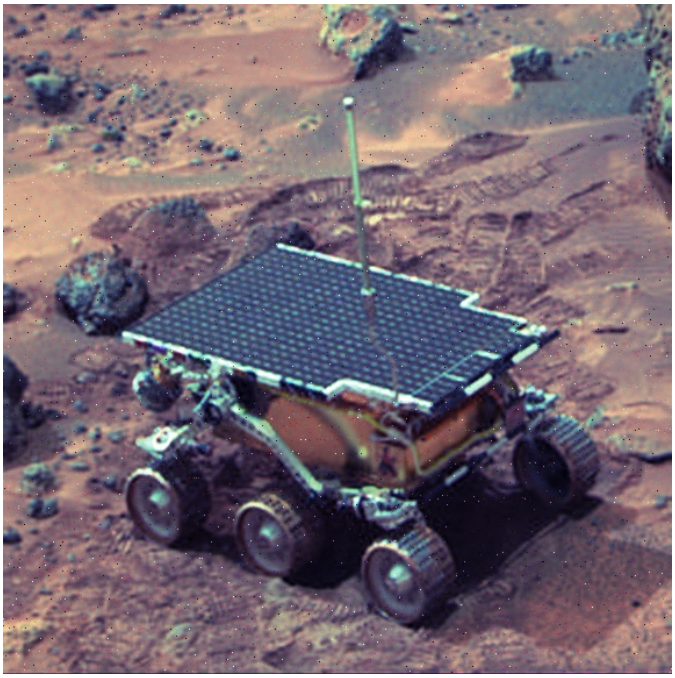


(B) Modified two-fold turbo scheme

Figure 7.28 Decoded HSL images with 2 iterations at E_b/N_o=1.0 dB

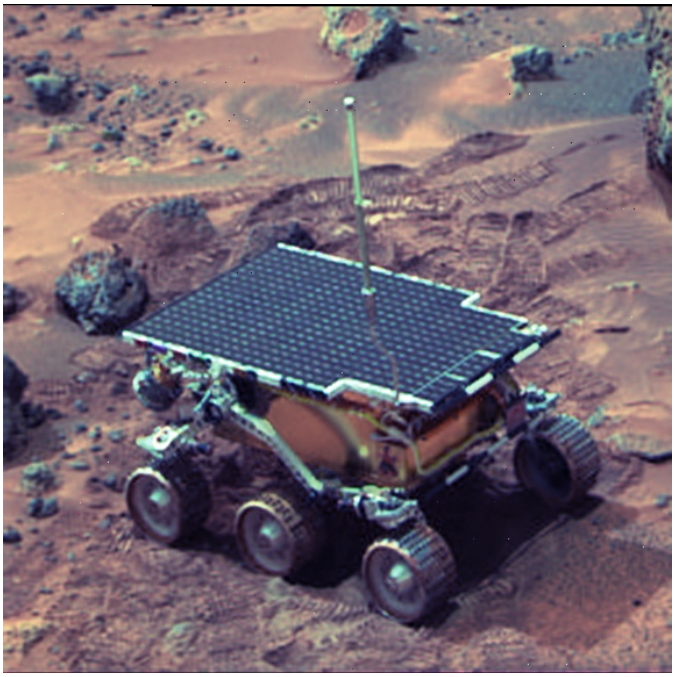


(A) Classical turbo scheme

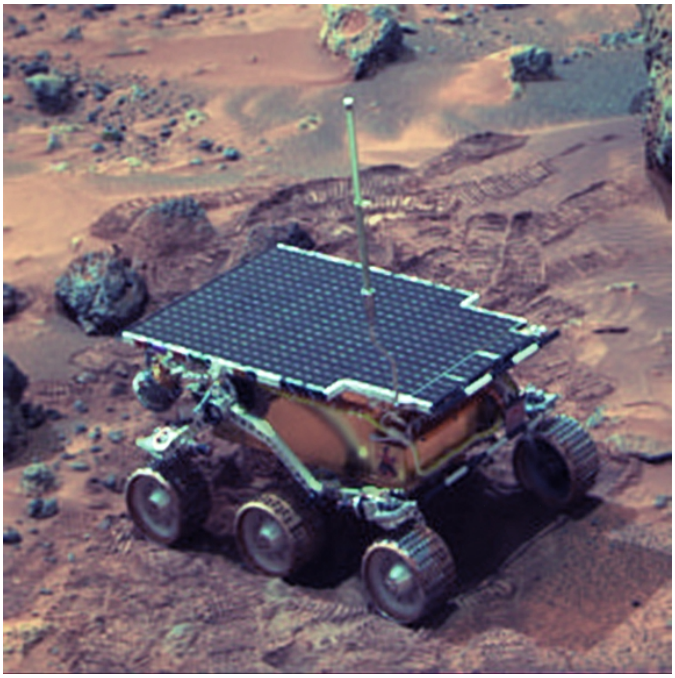


(B) Modified two-fold turbo scheme

Figure 7.29 Decoded HSL images with 4 iterations at E_b/N_o=1.0 dB

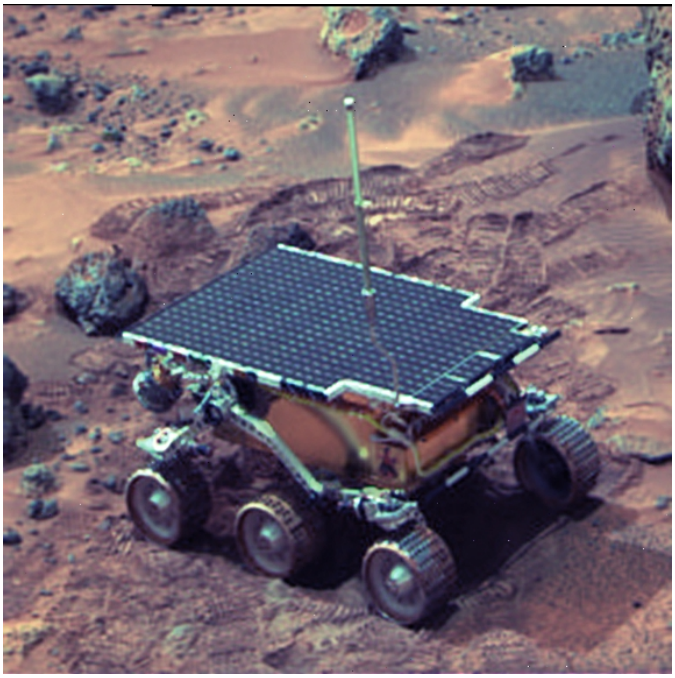


(A) Classical turbo scheme

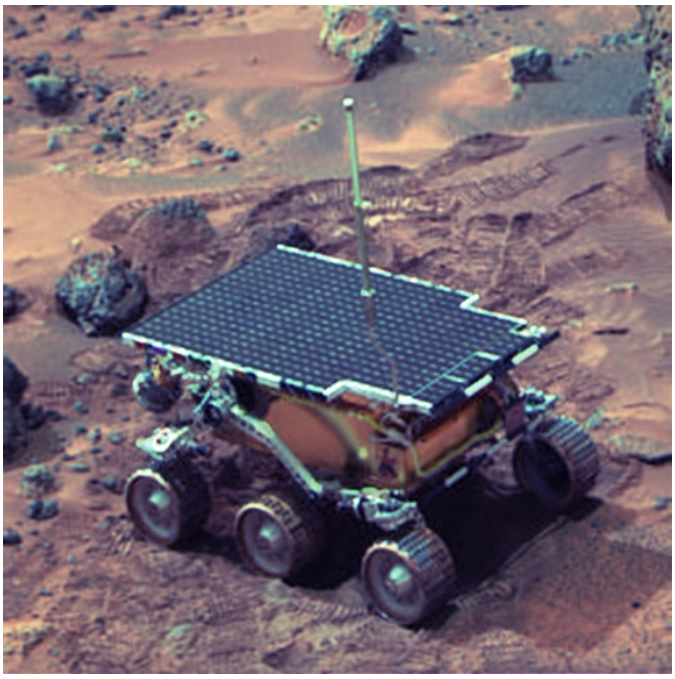


(B) Modified two-fold turbo scheme

Figure 7.30 Decoded HSL images with 8 iterations at E_b/N_o=1.0 dB



(A) Classical turbo scheme



(B) Modified two-fold turbo scheme

Figure 7.31 Decoded HSL images with 16 iterations at E_b/N_o=1.0 dB

In order to illustrate the PER improvement of the H, S and L components using the unequal two-fold turbo coding, Figure 7.32 has been included, where the *CL* and *MTF* schemes are compared. For the PER calculation, each HSL component has been

constructed as a separate image after turbo decoding, and has been compared to the corresponding colour component of the transmitted source image.

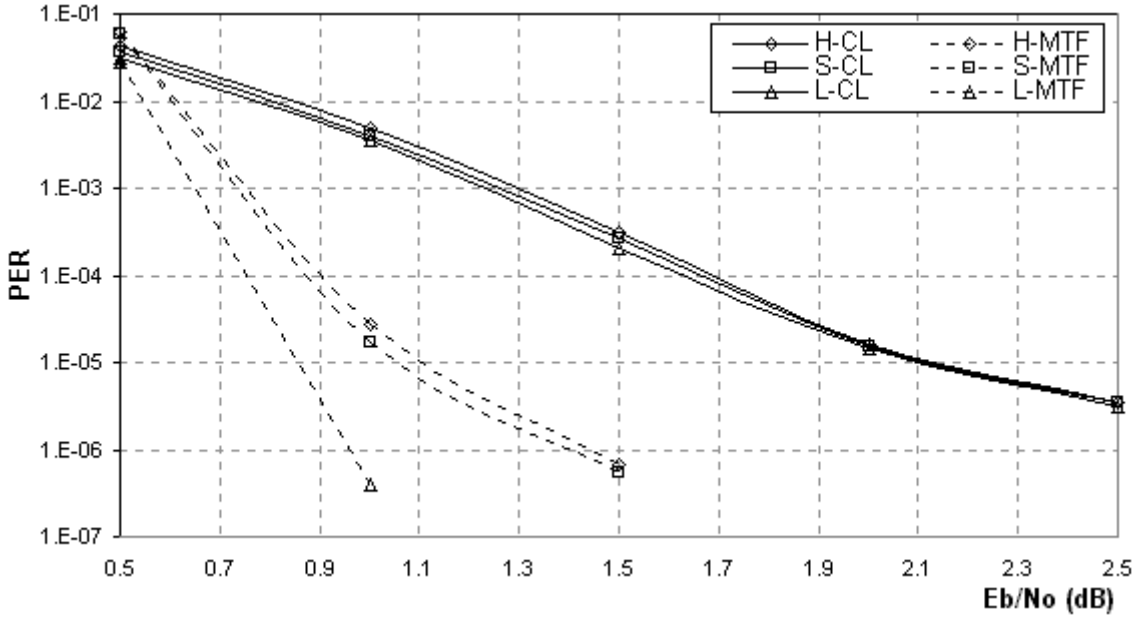
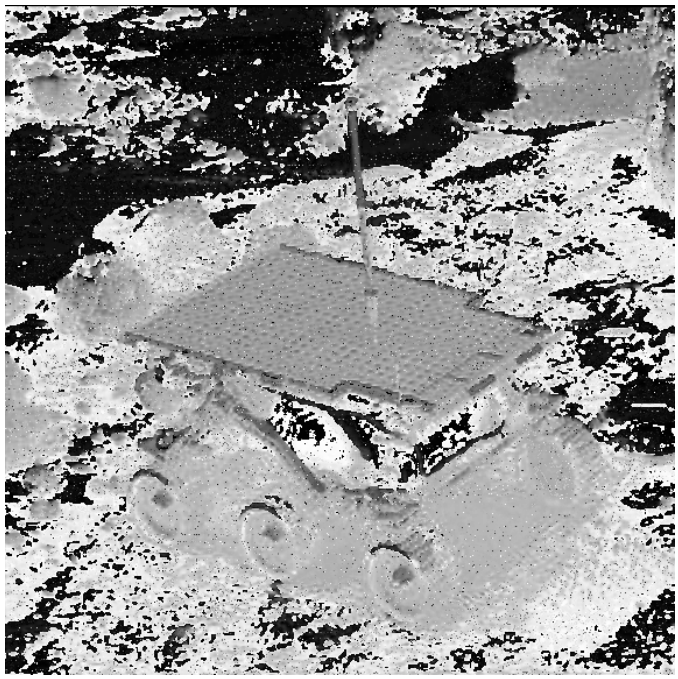


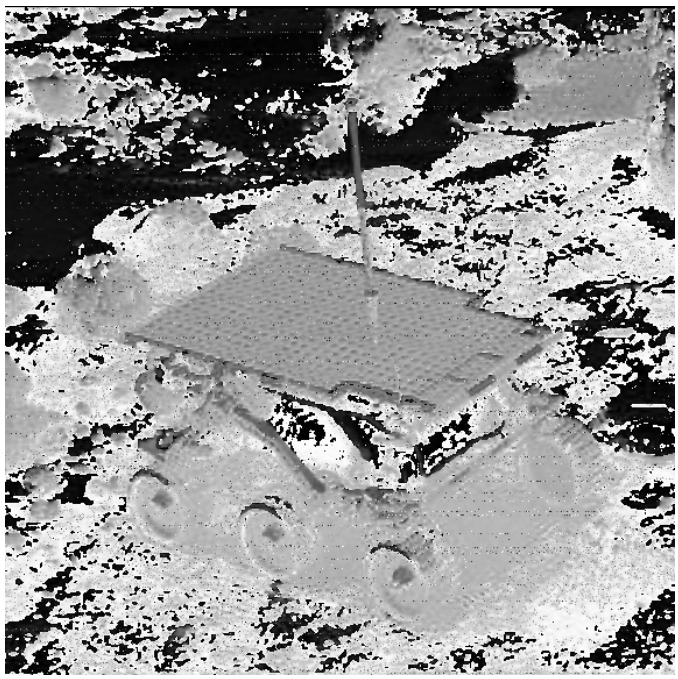
Figure 7.32 Pixel error performance of HSL components for 4 iterations

Figure 7.32 clearly shows the superior pixel error performance of the *MTF* over the *CL* scheme on a colour component basis. For the H and S components of the *MTF*, the 1.0 dB gain over the *CL* at PER=10^{-5} is due to the error performance improvement of the two-fold turbo codes over the classical turbo codes. However, the improved error performance in the case of the L component is a combined gain of the two-fold structure and the added error protection in the *MTF* scheme. Comparison of the L-*MTF* and the H-*MTF* and S-*MTF* curves shows that the unequal error protection scheme introduces an additional 0.2 dB gain to the two-fold turbo coding at PER=10^{-5}. More significantly, the L-*MTF* curve achieves about 1.2 dB gain over the L-*CL* curve at the same pixel error level.

In order to visually present the effects of PER improvement displayed in Figure 7.32, a set of decoded H, S and L images with 4 iterations are presented in Figure 7.33 through Figure 7.35. The *MTF* coded planes clearly exhibit a better picture quality than the *CL* coded planes due to the improved error performance of the two-fold turbo codes compared to the classical turbo codes. Another important point is that the luminance plane in Figure 7.35B has the best visual quality among the other decoded planes. As explained previously, this additional enhancement is a result of the unequal error protection stage introduced by the *MTF* scheme.

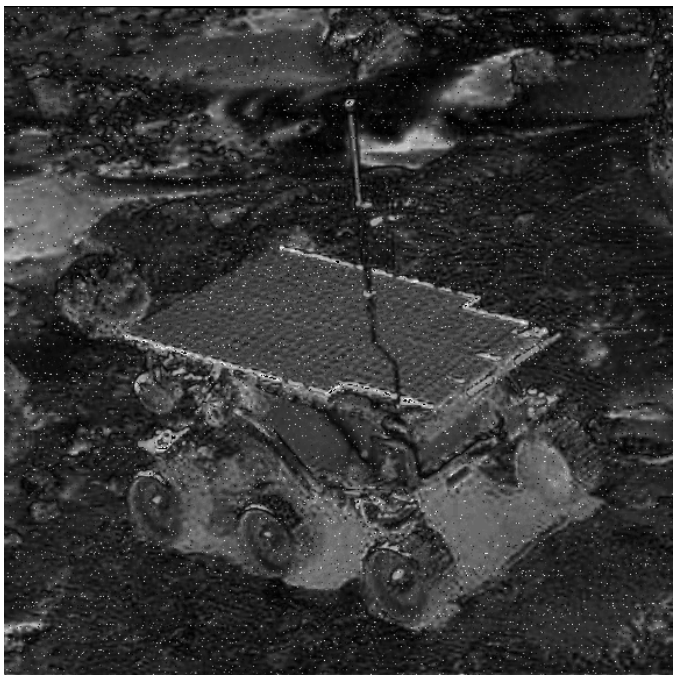


(A) Hue plane protected with the classical turbo scheme

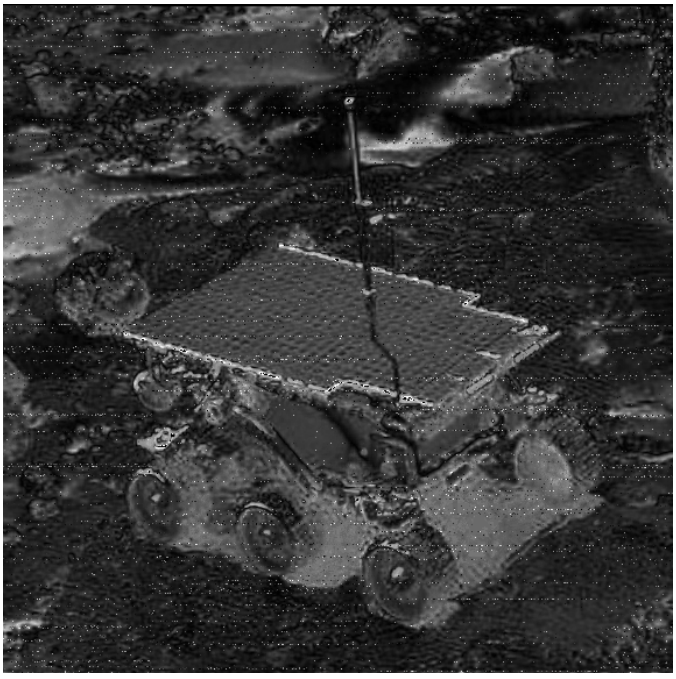


(B) Hue plane protected with the modified two-fold turbo scheme

Figure 7.33 Decoded hue components with 4 iterations at E_b/N_o=0.7 dB

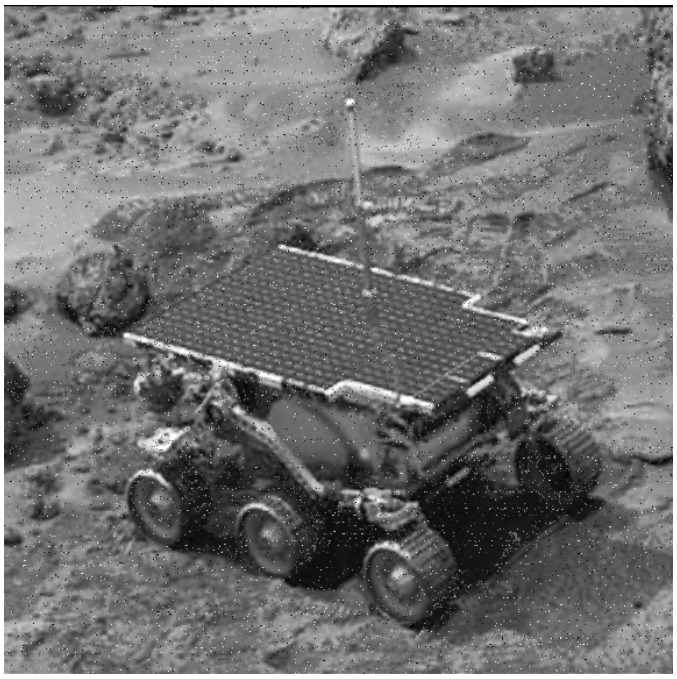


(A) Saturation plane protected with the classical turbo scheme

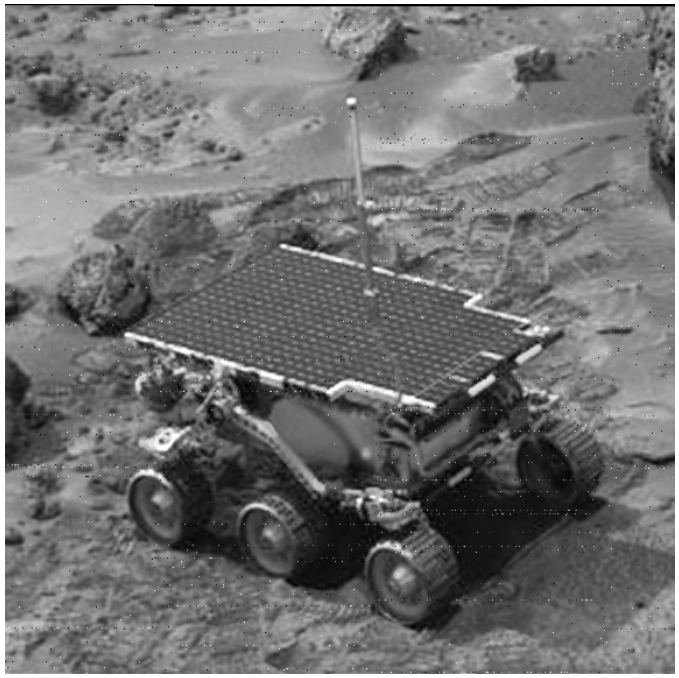


(B) Saturation plane protected with the modified two-fold turbo scheme

Figure 7.34 Decoded saturation components with 4 iterations at E_b/N_o=0.7 dB



(A) Luminance plane protected with the classical turbo scheme



(B) Luminance plane protected with the modified two-fold turbo scheme

Figure 7.35 Decoded luminance components with 4 iterations at E_b/N_o=0.7 dB

7.4. Discussion

In this chapter two turbo coding applications for reliable image transmission over noisy channels have been discussed.

In the first application, the turbo coded APEL transmission has been compared to the turbo coded JPEG and BMP image transmissions to demonstrate the advantages of APEL coding with respect to those well-known uncompressed and compressed image coding schemes. The error performance of the turbo coded image transmission schemes was evaluated over the AWGN as well as bursty channels.

Considering the AWGN channel, the performance of the turbo-JPEG scheme is found to be poor due to the inherent fragility of the JPEG structure and its inability to correct or restrict the propagation of any errors. The turbo-BMP scheme, on the other hand, performs well because of the complete independence of all pixels from one another. Hence, when the turbo decoder cannot repair certain bit errors, only pixels with the corrupted bits are affected. Finally, the turbo-APEL performance is comparable to the turbo-BMP scheme, even though it is compressed. More importantly, as the channel conditions improve, the turbo-APEL performs better than the turbo-BMP concatenation. This finding is explained by the effectiveness of the post-processing techniques used by the APEL decoder, which can correct pel errors using the correlation between the received bit planes.

The error performance for the turbo coded JPEG, BMP and the APEL coding schemes were also evaluated under the bursty Gaussian channel conditions. In this case, the pixel error performance of the JPEG scheme was found to be very poor compared to the BMP and the APEL schemes. In fact, for this channel, despite the turbo coding, JPEG images were so damaged that received image format could not even be decoded for visualisation. The two survivor schemes, namely the turbo-APEL and the turbo-BMP, were found to follow a similar trend to that observed for the Gaussian channel. As the turbo code was not designed to combat the burst errors, the pixel error curves for the two schemes were observed to be flatter than the ones obtained for the AWGN channel.

The psychometric studies for the bursty Gaussian channel also supported the pixel error evaluations. The crossover points on the psychometric PER plot (Figure 7.21) and the simulation PER plot (Figure 7.17) were surprisingly close in terms of E_b/N_o - a result that may verify the near-human visual assessment enabled by the pixel error rate calculation.

The second application of this chapter was a two-fold turbo based unequal error protection scheme, developed for the reliable transmission of HSL colour components of a digital still image in noisy channels. This application is an attractive option for transmitting the luminance information, to which the human eye is more sensitive, with heavier error protection level than the hue and the saturation components. The inherent sub-frame structure of the multifold coding introduced in Chapter 3 has been used for introducing a hierarchical coding strategy, which has only been discussed for two-fold turbo codes within the scope of this thesis.

Using the PER to quantify the visual noise, a classical turbo scheme performance was compared to the proposed unequal two-fold turbo code scheme. It has been shown that the decoder of the unequal protection scheme could provide up to 0.7 dB gain compared to the classical scheme at a pixel error rate of 10^{-5}. Furthermore, with the new decoding scheme, by performing 12 iterations less than the classical turbo decoder, it is possible to produce visually better HSL images, under certain channel conditions.

Chapter 8

Conclusions and Further Work

8. Conclusions and Further Work

8.1. Conclusions

The research presented in this thesis has covered a novel method of improving turbo codes' error performance as well as three low-complexity techniques to improve the iteration management, channel estimation and error detection aspects of iterative decoders. Also, two turbo code applications, which increase the reliability of compressed and uncompressed still image transmission over noisy channels, have been discussed. The work is aimed at introducing practical design concepts related to turbo codes and iterative decoding, which may be useful in software and hardware implementations of future error control and coding systems.

In the rest of this section, the partial contribution of each chapter to the research will be revisited briefly.

Chapter 3 has introduced the multifold turbo coding, which improves the error performance of traditional turbo codes in the waterfall region by using shorter interleaver lengths. Multifold encoding and decoding principles have been explained, and their differences from the classical turbo coding have been underlined. One class of the multifold turbo codes, namely the two-fold turbo codes, have been investigated, and their error performance, weight distribution and decoding complexity, have been evaluated for the g(7,5) and g(23,33) RSC component codes.

It has been shown that, for the g(7,5) code, coding gains up to 0.6 dB are possible with the parallel and serial decoding. In addition, for both parallel and serial decoding, fast convergence with less number of iterations could be achieved for the two-fold turbo codes.

The improved error performance of the two-fold turbo codes were also verified using the g(23,33) component code. Besides the 0.15 dB gain achieved with parallel decoding, the advantage of the two-fold turbo codes in terms of frame error performance was clearly observed. For 16 iterations, even though, the two-fold decoder's BER was worse than that of the classical decoder's, its frame error performance was better for both serial and parallel decoding. In fact, for parallel decoding, the FER of 4 two-fold iterations was even better than that achieved with 8 classical iterations.

The waterfall error performance improvement introduced by the multifold turbo codes is associated with the increased the randomness of the codeword weight distribution as well as the codeword weight. For the two-fold turbo code, the weight distribution can be made 2 to 5% more random for the g(23,33) and the g(7,5) RSC codes, respectively, which results in the coding gains mentioned above. An important insight gained from the weight analysis is that turbo codes with short interleaver lengths can perform better than those with longer interleaving degrees, provided that their weight properties are better.

Complexity analysis has shown that the two-fold decoders increase the classical turbo decoding latency by approximately 8.67%, which was found to be roughly the same for the g(7,5) and the g(23,33) codes. It is important to note that using multifold turbo codes increases the cost of decoding in terms of the number of computations, and the decoding complexity increases as the fold of the turbo codes is increased.

In Chapter 4, an iteration control mechanism that utilizes the soft output progress of the turbo decoder has been discussed.

It has been shown that the soft value sign change patterns during iterations- identified by the crossover coding algorithm- can be used to define a degree of certainty for each decoded information symbol. The number of high certainty codewords within a frame was found to be inversely proportional to the number of bit errors within that frame.
The crossover codeword feedback system, uses this inverse relationship to predict the number of bit errors after the completion of odd number of iterations, and triggers the decoder to perform further iterations unless the achieved BER is below a user-defined error performance threshold.

The iteration control system was observed to stabilize the BER performance while efficiently allocating the processing power available for the turbo decoder. Instead of blindly performing a constant number of iterations, the turbo decoder can externally be controlled by a more intelligent block that allows it to stop when necessary.
In addition, the crossover detector was found to increase the inherent turbo decoding latency by 0.56% to 2.71%, depending on the number of activations.

A channel estimation technique that combines pilot symbol insertion and decoder-based methods is introduced in Chapter 5.

The new channel estimation technique has been incorporated into a channel estimator block that performs channel initialisation as well as the noise variance update, derived from the turbo decoder's soft information output.

With the investigated estimation technique, although the channel information is provided to each frame based on the metrics obtained from previous frames, the turbo decoder still performs as it would in the presence of perfect channel information. In addition, the turbo decoder was observed to be almost unresponsive to channel estimation errors as long as they stay within a ±0.5 dB range.

It has also been shown that for AWGN channels, the channel estimation accuracy and stability depends on the filter length. In addition to that, due to the curve approximation errors, over and under estimation of the channel information is inevitable. However, when the sensitivity of the turbo decoder to channel mismatch is small, such estimation errors do not degrade the error performance of the decoder, under certain conditions.

Increasing the filter length in the channel estimator achieves stable noise variance estimation, at the price of longer response delays, which can potentially worsen the error performance when the channel noise varies fast with high power. Therefore, the filter length should be chosen according to the rate and peak power of noise variations in the communication channel.

The tracking capability of the new channel estimation technique has also been tested for AWGN, by varying the signal-to-noise ratio using sinusoidal functions. It has been shown that the new estimation technique can follow channel variance deviations of different rates, which makes this technique applicable to fading channels.

According to the complexity evaluation results, the channel estimator introduces, approximately, 6% increase in decoding delay during channel initialisation, and 1% after initialization, which is an important advantage from a practical point of view.

Chapter 6 covers the development of a novel soft-value derived frame error detection metric, as well as a hybrid ARQ scheme that can utilise one of four retransmission modes.

One important advantage of the frame error detection method is that it only introduces 1.56% additional computational complexity to the conventional operation of the turbo decoder.

The hybrid ARQ operation modes can be chosen according to required throughput and frame error performance in a communication system. According to the performance evaluation results for block sizes 512, 1000 and 2000, the detection algorithm has similar accuracy degrees. Retransmission of the primary information (modes 2 and 3) was observed to perform worse than transmitting the secondary information (modes 0 and 1) in terms of error performance and information throughput.

More importantly, retransmission of one set of punctured parity bits (mode 0) after a negative acknowledgement, achieves almost the same FER as the retransmission of all punctured parity bits (mode 1). Another advantage of using mode 0 is that its information throughput is higher than mode 1.

Two turbo coding applications, which increase the reliability of still image transmission over noisy channels, are given in Chapter 7.

The first application is based on the turbo coded APEL, JPEG and BMP image coding schemes. The error performance of the three schemes was evaluated over the random and bursty Gaussian channels.

In the AWGN channel, the JPEG transmission was found to perform the poorest due to its data structure. Even though the turbo coding stage can effectively minimize the number of bit errors, the visual impact of bit errors on JPEG images were observed to be quite severe depending on the location of the bit errors. The visual quality of the turbo coded BMP images, on the other hand, are the best because of the complete independence of all pixels from one another. When the turbo decoder cannot repair certain bit errors, only pixels with the corrupted bits are affected. The turbo coded APEL error performance is comparable to that of the turbo coded BMP scheme, even though APEL images are compressed. More importantly, as the channel conditions improve, the turbo-APEL performs better than the turbo-BMP concatenation. This finding shows that the post-processing techniques used by the APEL decoder can correct groups of pixels at a time using their correlation with the correctly decoded pixel groups.

The error performance for the turbo coded JPEG, BMP and the APEL coding schemes were also evaluated under the bursty Gaussian channel conditions. In this case, the pixel error performance of the JPEG scheme was found to be very poor compared to the BMP and the APEL schemes. In fact, for this channel, despite the turbo coding, JPEG images were so damaged that received image format could not even be recognized by the source decoder. The two survivor schemes, namely the turbo-APEL and the turbo-BMP, were found to follow a similar trend to that observed for the Gaussian channel. As the turbo code was not designed to combat the burst errors, the pixel error curves for the two schemes were observed to be flatter than the ones obtained for the AWGN channel.

The psychometric tests performed for the bursty Gaussian channel were found to be consistent with the pixel error evaluations.

The second application of in Chapter 7 was a multifold-fold turbo code based unequal error protection scheme, used for the reliable transmission of HSL colour components of a digital still image in noisy channel conditions. The sub-frame structure of the multifold coding has been used for providing two error protection levels. The stronger protection level is used for encoding the luminance information, to which the human eye is more sensitive, and the weaker level is used for coding the hue and the saturation components. In order to provide a benchmark for comparison, a classical turbo scheme with identical block length and similar code rate to the unequal error protection scheme was used for encoding the colour components with equal protection levels. Simulations have revealed that the decoder of the unequal scheme could provide up to 0.7 dB gain compared to the classical scheme at a pixel error rate of 10^{-5}. Moreover, with the new coding scheme, by performing 12 iterations less than the classical turbo scheme, it is possible to produce visually better HSL images, depending on the channel conditions.

8.2. Further Work

8.2.1. Unequal Error Protection Using Multifold Turbo Codes

In Chapter 3, the multifold turbo coding was introduced, and its superior error performance in comparison with the conventional turbo coding schemes was shown. Also, in Chapter 7 a simple UEP scheme that used a modified multifold coding technique with two protection levels was applied to the transmission of HSL colour components. This multifold-based UEP technique can be further investigated to provide higher number of protection levels,

and the research on this topic may provide useful results suitable for future turbo code applications.

In order to offer an insight into the proposed multifold UEP coding scheme, the generalized encoder block diagram is presented in Figure 8.1. The information frame is split into N_s equally sized segments, each of which constitute L_s bits. For the general unequal encoding case, N_s component encoders and N_s-*1* interleavers are used.

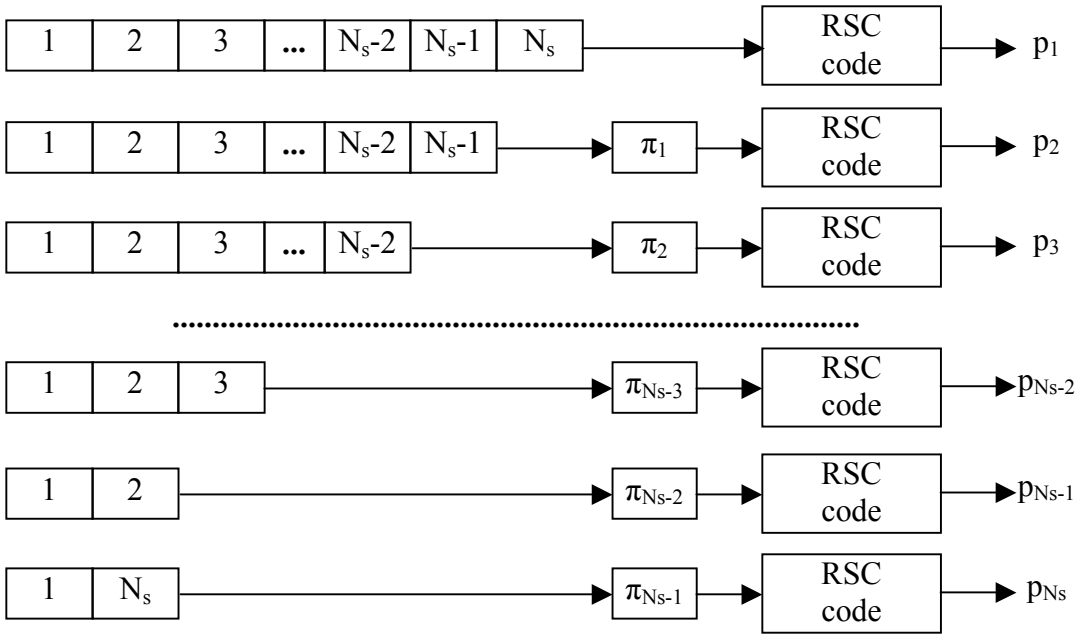


Figure 8.1 Generalized Multifold Turbo Encoder Designed for UEP

All segments are fed to the first component encoder to generate parity sequence p_1. Leaving one segment out at a time, the residual information segments are encoded by identical encoders superseding an interleaving stage until only one segment remains. As a final step, the last segment is combined with segment N_s and is encoded. If the code is systematic, the information frame is included as a part of the codeword $p_1\, p_2\, p_3 \ldots\, p_{Ns}$. The unpunctured code rate, R_{um}, for this turbo-coding scheme is a function of N_s (8.1). Notice that the encoding frequency for each segment is different except for segments N_s-*1* and N_s, which are encoded twice (recall that the soft information can be exchanged between minimum two component decoders during iterations).

$$R_{um} = \frac{2N_s}{N_s^{\,2} + 3N_s + 2} \qquad \textbf{(8.1)}$$

The encoding frequency per segment (f_e) can be calculated from (8.2), where S denotes the index of an information segment. Note that (8.2) is valid for all segments except segment N_s. For example, the f_e for the 2nd segment when N_s=*4* can easily be calculated as 3. In

addition, the unpunctured systematic code rate for the same example is equal to 4/15, as calculated from (8.2).

$$f_e = (N_S + 1) - S \qquad ,S \in Z^+,\ 1 \le S < N_s \qquad \textbf{(8.2)}$$

Note that for the proposed multifold UEP turbo coding scheme, using N_s segments provides N_s-*1* different levels of error protections. While the number of protection levels increases with N_s, the unpunctured code rate, R_{um}, decreases. Therefore, in order to obtain code rates higher than 1/3 with N_s>4, effective puncturing strategies need to be investigated for the proposed UEP scheme.

From an application point of view, this multifold UEP turbo scheme would be particularly suitable for compressed image [Xiang et al, 01] and speech [Burkert et al, 96] , [Caire & Lechner, 96] transmission, where short information block sizes are required. It is envisaged that incorporation of the soft output-based ARQ scheme discussed in Chapter 6 into the proposed UEP scheme, can improve the error performance of the coding system with a small increase in complexity. Enforcing the multifold-based UEP scheme with an efficient ARQ system seems to be an attractive option when operating with short information block lengths.

8.2.2. Combined Iteration Control and Channel Estimation

In Chapter 4 the sign changes of LLRs during iterations was used for developing metrics that were used for terminating the decoder iterations. Similarly in Chapter 5, the progress patterns of LLR magnitudes during turbo decoding were shown to provide useful information for updating the channel information.
As the channel estimation (CE) and the iteration control (IC) techniques are both soft value-derived, the two functions can be combined in one block, namely the CEIC block, which is external to the turbo decoder (Figure 8.2). The CEIC block will be capable of processing the LLRs of the turbo decoder after each iteration, and it will provide channel update information and an iteration flag, which prompts the turbo decoder to either stop or to continue iterating.

The sign progress and magnitude progress of the LLRs are the two criteria that are worth considering while developing the metrics used by the CEIC block. The channel dependence of the soft information progress patterns were previously addressed in sections 4.3 and 5.3. It is also known that there is a correlation between the number of decoder

iterations and the channel conditions; as the noise level decreases, the decoder needs to perform less iterations to achieve a given BER.

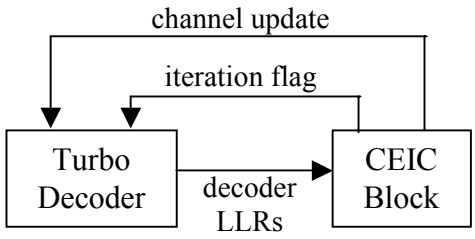


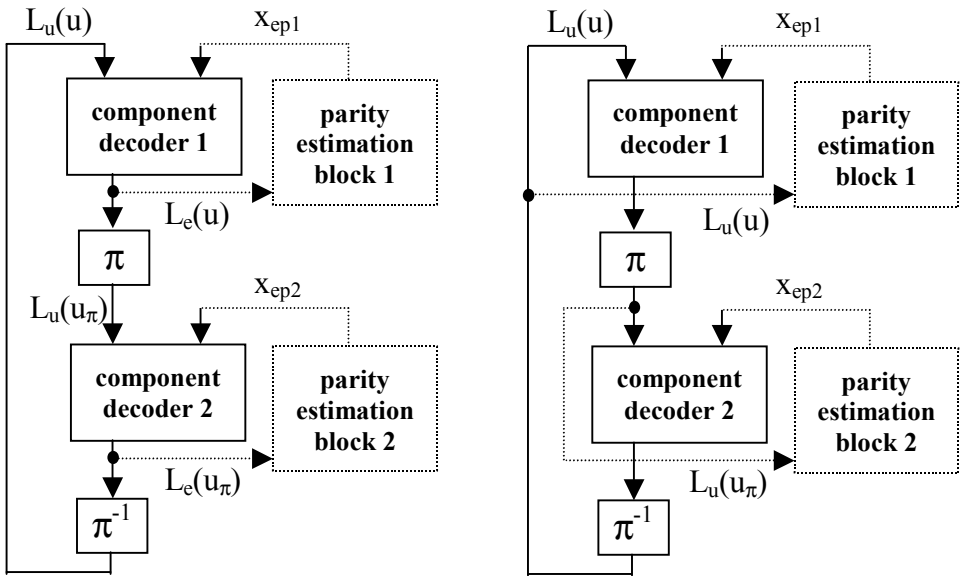
Figure 8.2 Combined channel estimation and iteration control for turbo decoders

Therefore, it is envisaged that both the sign and the magnitude progress patterns of the decoder LLRs can either be used individually or jointly to develop the estimation metrics used by the CEIC block. The methods of developing such metrics were discussed in Chapters 4 and 5, and can easily be applied to the proposed CEIC block.

8.2.3. Iterative Decoding with Parity Estimation

Iterative decoding inherently improves the confidence level of information symbols, and attempts to converge to the maximum likelihood solution for each information symbol. During decoding iterations, the received parity information is left unchanged and is used for improving the confidence level of the received information symbols. However, if the improved reliability of the decoded information symbols could be reflected to the parity symbols, it would be possible to reduce the number of decoding iterations without any loss in error performance.

To achieve this, two different approaches to parity estimation using the soft value progress of the information symbols are proposed (Figure 8.3). In both schemes there is one parity estimation block that is associated with a component decoder. The difference between the two configurations is the type of soft information that is fed to the parity estimators.



configuration 1 *configuration 2*

Figure 8.3 Parity estimation configurations for turbo decoders

In configuration 1, the first parity estimator accepts the extrinsic information from the first component decoder, and delivers the estimated parity information, x_{ep1} to the same decoder. The second parity estimation block operates in a similar way, processing the extrinsic information of the second component decoder.

Parity estimation in configuration 2 is based on exchanging the input information to the two parity estimators, making sure that the input to the first estimator is de-interleaved whereas the input to the second estimator is interleaved. Estimated parity information is fed back to the component decoders as in configuration 1.

Currently, the performance of the two proposed parity estimation configurations is not known. Also, in order to develop an effective parity estimation technique, there are three important issues that need to be mentioned, which are as follows.

1)- Interpretation of the extrinsic information: The primary function of the parity estimator is to accept the soft decoder information, related to the information symbols, and to extract useful parity information from it. One way of doing this might be to use the survivor path in the decoding trellis, similar to the Viterbi algorithm [Lin & Costello, 83] , and to use the parity bits on the trellis branches.

2)- Type of estimated parity information: Another issue that needs to be thought about is whether to feed hard or soft parity information to the component decoders. If hard values are preferred, the branch values obtained from the survivor path might be used directly.

Using soft parity information can be more problematic in the sense that a method of mapping the soft extrinsic information to soft parity information would have to be devised.

3)- Incorporation and update of estimated parity: The final problem that needs to be resolved is how the parity information will be incorporated into the component decoder and how it will be updated during iterations. One intuitive approach might be to gradually replace the received parity symbols with their estimates during iterations. The replaced parity bits would be associated with the ones associated with the information symbols that are decoded with high confidence levels. Consequently, as the iterations proceed, the number of replaced parity symbols would increase at the same rate as the number of information symbols whose confidence levels increase during iterations.

References

References

[Acton, 70] Acton F. S., "*Numerical Methods That Work*", New York, Harper & Row, 1970.

[Bahl et al, 74] Bahl L.R., Cocke J., Jelinek F., Raviv J., "*Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate*", IEEE Transactions on Information Theory, Vol. IT-20, March 1974, pp.284-287.

[Battail, 98] Battail G., "*A Conceptual Framework for Understanding Turbo Codes*", IEEE J. Select. Areas Commun., Vol. 16, No. 2, Feb. 1998, pp. 245-254.

[Benedetto & Montorsi, 95] Benedetto S., Montorsi G., "*Role of Recursive Convolutional Codes in Turbo Codes*", IEE. Electronics Letters, Vol.31, No.11, May 1995, pp.858-859.

[Berrou & Glavieux, 96] Berrou C., Glavieux A., "*Near Optimum Error Correcting Coding and Decoding: Turbo-Codes*", IEEE Transactions on Communications, Vol. 44, No. 10, October 1996, pp. 1261-1271.

[Berrou et al, 93] Berrou C., Glavieux A., Thitimajshima P.,"*Near Shannon Limit Error-correcting Coding and Decoding: turbo-codes*", Proc. IEEE International Conference on Communications, Geneva, Switzerland, May 1993, pp. 1064-1070.

[Biglieri & Volski, 94] Biglieri E., Volski V., "*The weight distribution of the iterated product of single-parity-check codes is approximately Gaussian*", IEE. Electronics Letters, Vol. 30, No. 12, June 1994, pp. 923-924.

[Buckley & Wicker, 00] Buckley M. E., Wicker S. B., "*The Design and Performance of a Neural Network for Predicting Turbo Decoding Error with Application to Hybrid ARQ Protocols*", IEEE Transactions on Communications, Vol. 48, No. 4, April 2000, pp. 566-576.

[Buckley & Wicker, 99] Buckley M. E., Wicker S. B., "*A Neural Network for Predicting Decoder Error in Turbo Decoders*", IEEE Communications Letters, Vol. 3, No. 5, May 1999, pp. 145-147.

[Burkert et al, 96] Burkert F., Caire G., Hagenauer J., Hindelang T., Lechner G., "*Turbo Decoding with Unequal Error Protection Applied to GSM Speech Coding*", Proc. IEEE GLOBECOM 1996, November 1996.

[Caire & Lechner, 96] Caire G., Lechner G., "*Turbo Codes with Unequal Error Protection*", IEE. Electronics Letters, Vol. 32, No. 7, March 1996, pp. 629-631.

[Carlson, 86] Carlson A.B., "*Communication Systems*", USA, McGraw-Hill Series, 1986.

[CCSDS, 98] CCSDS Recommendation for Telemetry Channel Coding – Rev. 4 pp. 4-2,3, May 1998.

[Chan & Gerantiotis, 97] Chan W.C., Gerantiotis E., "*An Adaptive Hybrid FEC/ARQ Protocol Using Turbo Codes*", Proc. IEEE International Conference of Universal Personal Communications, Vol. 2, 1997, pp. 541-545.

[Chase & Bown, 86] Chase W., Bown F., "*General Statistics*", USA, John Wiley & Sons, 1986.

[Chass & Gubeskys, 00] Chass A., Gubeskys A., "*On Performance/COmplexity Analysis and SW Implementation of Turbo Decoding*", Proc. 2nd International Symposium on Turbo Codes & Related Topics, Brest, France, 2000, pp. 531-534.

[Chippendale et al, 99a] Chippendale P., Tanriover C., Honary B., "*Turbo Coded Image Transmission over Noisy Channels*", Fourth Volume on Communication Theory and Applications, Research Studies Press Ltd., 2000, UK, pp 277-286.

[Chippendale et al, 99b] Chippendale P., Tanriover C., Honary B., "*Enhanced Image Coding for Noisy Channels*", Proc. 7th IMA International Conference, Cirencester, UK, December 1999, pp 94-103.

[Chippendale et al, 99c] Chippendale P., Honary B., Arthur P., Maundrell M., "*Data Encoding System*", International Patent, Ref.: PCT GB 98/01877, 1999.

[Chippendale, 98] Chippendale P., "*Transmission of Images over Time-Varying Channels*", PhD Thesis, Lancaster University, UK, August 1998.

[CIE, 00] CIE, "*General information on the International Commission on Illumination*", http://www.cie.co.at/cie, International Commission on Illumination Home Page, 2000.

[Cooper & McGillem, 88] Cooper G. R., McGillem C.D., "*Modern Communications and Spread Spectrum*", Singapore, McGraw-Hill, 1988.

[Costello et al, 98] Costello D.J.Jr., Hagenauer J., Imai H., Wicker S.B., "*Applications of Error-Control Coding*", IEEE Transactions on Information Theory, Vol.44, No.6, October 1998, pp.2531-2560.

[Coulton & Honary, 98] Coulton P., Honary B., "*Providing Channel Information to Turbo Decoders*", Proc. International Conference on Telecommunications, Vol. 3, Aug. 1998, pp. 59-63.

[Coulton et al, 00] Coulton P., Tanriover C., Wright B., Honary B., "*Simple Hybrid Type II ARQ Technique Using Soft Output Information*", IEE. Electronics Letters, Vol.36, No.20, September 2000, pp.1716-1717.

[Divsalar & Pollara, 95] Divsalar D., Pollara F., "*Turbo Codes for Deep-Space Communications*", TDA Progress Report 42-120, 15 February 1995.

[Drury et al., 01] Drury G., Markarian G., Pickavance K., "*Coding and Modulation for Digital Television*", USA, Kluwer Academic Publishers, 2001.

[Eroz & Fuja, 98] Eroz M., Fuja T. E., "*A Hybrid-ARQ System Using Rate-Compatible Trellis Codes Designed for Rayleigh Fading*", European Transactions on Telecommunications & Related Technologies, Vol.9, No.6, December 1998, pp.483-495.

[Frey & MacKay, 00] Frey B.J., MacKay D.J.C., "*Irregular Turbo-Like Codes*", Proc. 2nd International Symposium on Turbo Codes & Related Topics, Brest, France, 2000, pp. 67-72.

[Gross & Gulak, 98] Gross W.J., Gulak P.G., "*Simplified MAP Algorithm Suitable for Implementation of Turbo Decoders*", IEE Electronics Letters, Vol.34, No.16, August 1998, pp.1577-1578.

[Hagenauer et al, 96] Hagenauer J., Offer E., Papke L., "*Iterative Decoding of Binary Block and Convolutional Codes*", IEEE Transactions on Information Theory, Vol.42, No.2, March 1996, pp.429-45.

[Hagenauer & Hoeher, 89] Hagenauer J., Hoeher P., "*A Viterbi Algorithm with Soft-Decision Outputs and its Applications*", Proc. IEEE GLOBECOM 1989, November 1989, pp. 1680-1686.

[Han & Takeshita, 01] Han J., Takeshita O. Y., "*On the Decoding Structure for Multiple Turbo Codes*", Proc. IEEE. International Symposium on Information Theory, June 2001, p 98.

[Haykin, 94] Haykin S., "*Communication Systems*", Third Edition, USA, John Wiley & Sons, 1994.

[Heegard & Wicker, 99] Heegard C., Wicker S.B., "*Turbo Coding*", USA, Kluwer Academic Publishers, 1999.

[Hoeher, 97] Hoeher P., "*New Iterative ("Turbo") Decoding Algorithms"*, Proc. International Symposium on Turbo Codes, Brest, France, 1997, pp. 63-70.

[Jordan & Michols, 96] Jordan M., Michols R., "*The Effects of Channel Characterization on Turbo Code Performance*", Proc. MILCOM '96, McLean, VA, October 1996, pp.17-21.

[Kim & Bahk, 96] Kim Y., Bahk S., "*An Adaptive Hybrid ARQ Scheme Using Shortened Codes*", Proc. IEEE GLOBECOM 1996, part vol.3, 1996, pp.2157-61.

[Kim & Yoon, 00] Kim J.W., Yoon W.S., "*Turbo Code with Iterative Channel Estimator Using Soft-Output of Turbo Decoder*", IEE Electronics Letters, Vol.36, No.18, August 2000, pp.1560-1562.

[Lin et al, 84] Lin S., Costello D.J. Jr., Miller M.J., "*Automatic-Repeat Request Error-Control Schemes*", IEEE Communications Magazine, Vol. 22, No. 12, December 1984, pp. 5-17.

[Lin & Costello, 83] Lin S., Costello D.J.Jr., "*Error Control Coding: Fundamentals and Applications*", USA, Prentice-Hall, 1983.

[Liu et al, 00] Liu Y., Lin S., Fossorier M.P.C., "*MAP Algorithms for Decoding Linear Block Codes Based on Sectionalized Trellis Diagrams*", IEEE Transactions on Communications, Vol.48, No.4, April 2000, pp. 577-587.

[Liu & Zarki, 97] Liu H., Zarki M.E., "*Performance of H.263 Video Transmission over Wireless Channels Using Hybrid ARQ*", IEEE Journal on Selected Areas in Communications, Vol. 15, No. 9, December 1997, pp.1775-1786.

[Massey, 92] Massey J.L., "*Deep-Space Communications and Coding: A Marriage Made in Heaven*", Advanced Methods for Satellite and Deep Space Communications, Lecture Notes in Control and Information Sciences, No. 182, pages 1-17, Heidelberg and New York, Springer, 1992.

[Massey & Costello, 00] Massey P.C., Costello D.J., Jr., "*New Development in Asymmetric Turbo Codes*", Proc. 2nd International Symposium on Turbo Codes & Related Topics, Brest, France, 2000, pp. 93-100.

[Matache et al, 00] Matache A., Dolinar S., Pollara F., "*Stopping Rules for Turbo Decoders*", TMO Progress Report 42-142, 15 August 2000.

[Miller, 96] Miller G.M., "*Modern Electronic Communication*", Fifth Edition, Prentice-Hall Inc. USA, 1996.

[Otung, 01] Otung I., "*Communication Engineering Principles*", Great Britain, Palgrave, 2001.

[Pietrobon, 98] Pietrobon S.S., "*Implementation and Performance of a Turbo/MAP Decoder*", International Journal of Satellite Communications, Vol. 16, January/February, 1998.

[Pyndiah, 98] Pyndiah R.M., "*Near-Optimum Decoding of Product Codes: Block Turbo Codes*", IEEE Transactions on Communications, Vol.46, No.8, August 1998, pp. 1003-1010.

[Rasmussen & Wicker, 94] Rasmussen L. K., Wicker S. B., "*The Performance of Type-I Trellis Coded Hybrid-ARQ Protocols over AWGN and Slowly Fading Channels*", IEEE Transactions on Information Theory, Vol. 40, No. 2, March 1994, pp. 418-428.

[Reed & Asenstorfer, 97] Reed M., Asenstorfer J., "*A Novel Variance Estimator for Turbo Code Decoding*", Proc. International Conference of Telecommunications, Melbourne, Australia, April 1997, pp. 173-178.

[Reid et al, 01] Reid A. C., Gulliver T. A., Taylor D. P., "*Convergence and Errors in Turbo-Decoding*", Proc. IEEE International Symposium on Information Theory, Washington, D.C., U.S.A., June 2001, p. 26.

[Robertson et al, 95] Robertson P., Villebrun E., Hoeher P., "*A Comparison of Optimal and Sub-Optimal MAP Decoding Algorithms Operating in the Log Domain*", Proc. International

Conference in Communications `95, Vol. 2, 1995, pp. 1009-1013.

[Rosmansyah et al, 00] Rosmansyah Y., Sweeney P., Sweeting M. N., "*Efficient Turbo Coded ARQ for Low Earth Orbit Microsatellites*", Fourth Volume on Communication Theory and Applications, Research Studies Press Ltd., 2000, UK, pp 325-329.

[Ryan, 97] Ryan W.E., "*A Turbo Code Tutorial*", submitted to GLOBECOM 97, New Mexico State University, USA, 1997.

[Shannon, 48] C.E. Shannon, "*A Mathematical Theory of Communication*", Bell Systems Technical Journal, Vol.27, July and October 1948, pp.379-423 and 623-656.

[Shao et al, 99] Shao R.Y., Lin S., Fossorier M.P.C., "*Two Simple Stopping Criteria for Turbo Decoding*", IEEE Transactions on Communications, Vol.47, No.8, August 1999, pp.1117-1120.

[Shibutani et al, 99] Shibutani A., Suda H., Adachi F., "*Reducing Average Number of Turbo Decoding Iterations*", IEE Electronics Letters, Vol.35, No.9, April 1999, pp.701-702.

[Shin et al, 00] Shin H., Kim S., Lee J.H., "*Turbo Decoding in a Rayleigh Fading Channel with Estimated Channel State Information*", Proc. IEEE VTC2000, Boston, MA, USA, September 2000.

[Singh, 67] Singh J., "*Great Ideas in Information Theory, Language and Cybernetics*", London, Constable & Company, 1967.

[Sklar, 97] Sklar B., "*A Primer on Turbo Code Concepts*", IEEE Communications Magazine, Vol.35, No.12, December 1997, pp. 94-102.

[Sklar, 88] Sklar B., "*Digital Communications, Fundamentals and Applications*", New Jersey, Prentice Hall, 1988.

[Summers & Wilson, 98] Summers T.A., Wilson S.G., "*SNR Mismatch and Online Estimation in Turbo Decoding*", IEEE Transactions on Communications, Vol.46, No.4, April 1998, pp. 421-423.

[Svirid, 95] Svirid Y.V., "*Weight Distributions of Turbo-Codes*", Proc. IEEE. International Symposium on Information Theory, September 1995, p 38.

[Takeshita et al, 99] Takeshita O.Y., Collins O.M., Massey P.C., Costello D.J, Jr., "*A Note on Asymmetric Turbo-Codes*", IEEE

Communications Letters, Vol. 3, No. 3, March 1999, pp. 69-71.

[Tanaka & Sakakibara, 96] Tanaka H., and Sakakibara K., "*Performance of Reed-Solomon Coded Type-I Hybrid ARQ Scheme on Fading Channels*", Proc. IEEE GLOBECOM 1996, part vol.3, 1996, pp.2148-52.

[Tanriover & Honary, 02a] Tanriover C., Honary B., "*Image Transmission Using Unequal Error Protection with Two-fold Turbo Codes*", submitted to IEEE International Telecommunications Symposium, Natal, Brazil, September 2002.

[Tanriover & Honary, 02b] Tanriover C., Honary B., "*Crossover Codeword Algorithm for Iteration Control*", submitted to IEEE Semiannual Vehicular Technology Conference, Vancouver, Canada, September 2002.

[Tanriover et al, 01a] Tanriover C., Xu J., Lin S., Honary B., "*Improving Turbo Code Error Performance by Multifold Coding*", scheduled to appear in IEEE Communications Letters, May 2002.

[Tanriover et al, 01b] Tanriover C., Xu J., Lin S., Honary B., "*Multifold Turbo Codes*", Proc. IEEE. International Symposium on Information Theory, June 2001, p 145.

[Tanriover & Honary, 00a] Tanriover C., Honary B., "*Modified Statistical Tracking System*", Internal report, Department of Communication Systems, Lancaster University, February 2000, ref:LCRC\NDS03\02.2000\03.

[Tanriover & Honary, 00b] Tanriover C., Honary B., "*Recent Work on Modified Statistical Tracking System*", Internal report, Department of Communication Systems, Lancaster University, March 2000, ref:LCRC\NDS04\03.2000\04.

[Tanriover & Honary, 00c] Tanriover C., Honary B., "*An Enhanced Decision Stage for Turbo Codecs-Statistical Tracking System (STS)*", 1st Annual PostGraduate Symposium on the Convergence of Telecommunications. Networking and Broadcasting, Liverpool, UK, June 2000, pp 257-262.

[Tanriover, 00d] Tanriover C., "*Crossover Codeword Error Detection Algorithm*", 1st Student Conference on Communication

Systems and Applications, Lancaster, UK, July 2000, pp 53-58.

[Tanriover et al, 99a] Tanriover C., Chippendale P., Honary B., "*Turbo Code Application to Image Transmission*", IEE Colloquium, Turbo Codes in Digital Broadcasting-Could it Double Capacity?, London, UK, November 1999, pp 17/1-17/6.

[Tanriover & Honary, 99b] Tanriover C., Honary B., "*Statistical Tracking System*", Internal report, Department of Communication Systems, Lancaster University, June 1999, ref:LCRC\NDS02\06.1999\02.

[Taub & Schilling, 86] Taub H., Schilling D.L., "*Principles of Communication Systems*", Second Edition, Singapore, McGraw-Hill, 1986.

[TenBrink, 00] TenBrink S., "*Rate One-Half Code for Approaching the Shannon Limit by 0.1 dB*", IEE Electronics Letters, Vol.36, No.15, July 2000, pp.1293-1294.

[Valenti, 96] Valenti M.C., "*An Introduction to Turbo Codes*", Internal Report, Virginia Tech, USA, May 1996.

[Valenti & Woerner, 98] Valenti M.C., Woerner B.D., "*Refined Channel Estimation for Coherent Detection of Turbo Codes over Flat-Fading Channels*", IEE Electronics Letters, Vol.34, No.17, August 1998, pp.1648-1649.

[Valenti & Woerner, 99] Valenti M.C., Woerner B.D., "*A Bandwidth Efficient Pilot Symbol Technique for Coherent Detection of Turbo Codes over Fading Channels*", IEEE Military Communications Conference (MILCOM `99), Atlantic City, NJ, November 1999.

[Viterbi, 98] Viterbi A.J., "*An Intuitive Justification and a Simplified Implementation of the MAP Decoder for Convolutional Codes*", IEEE Journal on Selected Areas in Communications, Vol. 16, No. 2, February 1998, pp. 260-264.

[Vivier, 01] Vivier G., "*Simulations of HARQ for Turbo-Coded Services in UMTS-TDD*", 4th European Personal Mobile Communications Conference, Vienna, Austria, February 2001.

[Wallace, 91] Wallace G.K., "*The JPEG Still Picture Compression Standard*", Communications of the ACM, USA, April 1991, Vol.34, No.4, pp. 30-44.

[Wicker, 95] Wicker S.B., "*Error Control Systems for Digital Communication and Storage*", USA, Prentice-Hall, 1995.

[Xiang et al, 01] Xiang W., Barbulescu S.A., Pietrobon S.S., "*Unequal Error Protection Applied to JPEG Image Transmission Using Turbo Codes*", Proc. IEEE Information Theory Workshop, September 2001, pp. 64-66.

[Zhai & Fair, 01] Zhai F., Fair I. J., "*Improved Performance of Error Detection in Turbo Decoding by Incorporating a Short CRC with the Mean-Sign-Change Criterion*", Proc. IEEE International Symposium on Information Theory, Washington, D.C., U.S.A., June 2001, p. 143.

List of Abbreviations

List of Abbreviations

A	Amplitude
ACK	positive ACKnowledgement
APEL	Absolute Picture ELement
ARPANET	Advanced Research Project Agency Network
ARQ	Automatic Repeat reQuest
AWGN	Additive White Gaussian Noise
BCJR	Bahl Cocke Jelinek Raviv
BMP	BitMaP
BPC	Bit Plane Coding
BPSK	Binary Phase Shift Keying
BER	Bit Error Rate
CCG	Crossover Codeword Generator
CCSDS	Consultative Committee for Space Data Systems
cd	Component Decoder
CD	Compact Disc
CHEST	CHannel ESTimation
CIE	International Commission on Illumination
cmp	CoMParator
CRC	Cyclic Redundancy Check
DDM	Decoder Derived Method
DEDN	Decoder Error Detecting Network
dmx	DeMultipleXer
FEC	Forward Error Correction
FED	Frame Error Detector
FEDN	Future Error Detecting Network
FER	Frame Error Rate
FIFO	First In First Out
fpo	Floating Point Operation
GBN	Go Back N
GR	Gap Ratio
GSM	
hd	Hard Decision
HDA	Hard Decision Aided
HSL	Hue Saturation Luminance
IC	Integrated Circuit or Iteration Control
IIR	Infinite Impulse Response
LLR	Log Likelihood Ratio
LSB	Least Significant Bit
JPEG	Joint Photographic Experts Group
MAP	Maximum A Posteriori
MSB	Most Significant Bit
mx	MultipleXer
NACK	Negative ACKnowledgement
pdf	Probability Density Function
pel	Picture ELement
PER	Pixel Error Rate
PSI	Pilot Symbol Insertion
PWG	Progress Word Generator
rms	Root Mean Square
RS	Reed Solomon
RSC	Recursive Systematic Convolutional

SAW	Stop And Wait
SCR	Sign Change Ratio
SISO	Soft In - Soft Out
SNR	Signal-to-Noise Ratio
SOVA	Soft Output Viterbi Algorithm
SPC	Single Parity Check
SR	Selective Repeat
UEP	Unequal Error Protection
UMTS	Universal Mobile Telecommunication System
WLL	Wireless Local Loop

List of Symbols

List of Symbols

$\mathbf{a_k}$	Message bit k
$\boldsymbol{\alpha}$	Forward recursion state transition probability
$\boldsymbol{\beta}$	Backward recursion state transition probability
B	Bandwidth
$\mathbf{b_k}$	Parity bit k
C	Channel capacity
C(*a*,*b*)	Number of combinations in *a* taken *b* at a time
$\mathbf{c_k}$	Code bit k
$\boldsymbol{\sigma}$	Standard deviation of a random variable
$\boldsymbol{\mu}$	Average of a set of variables
$\boldsymbol{\Delta}$	Relative error
$\mathbf{C_s}$	Total number of information sub-frames
$\mathbf{C}_x$	Complexity of block *x*
$\mathbf{CW_H}$	Crossover codeWord set of High certainty
$\mathbf{CW}_{k,\mathbf{H}}$	Crossover codeWord set of High certainty that corresponds to information symbol *k*
$\mathbf{CW}_k$	Crossover codeWord of information symbol *k*
$\boldsymbol{\delta}$	Dirac function
$\mathbf{d_{min}}$	Minimum distance
E	Energy
$\mathbf{E_b}$	Energy per bit
erf	Error function
erfc	Complementary error function
$\mathbf{E_s}$	Energy per modulated symbol
$\mathbf{E_t}$	Error threshold
$\boldsymbol{\Phi}$	Frame error detection metric
$\boldsymbol{\Phi_e}$	Frame error detection metric for erroneous frames
$\boldsymbol{\Phi_{ef}}$	Frame error detection metric for error free frames
$\boldsymbol{\Phi_t}$	Frame error detection metric
f(.)	Function representation
$\mathbf{F_I}$	Iteration stop flag
$\mathbf{f_n}$	Channel initialisation function
$\mathbf{f_\sigma}$	Noise estimation function
$\boldsymbol{\gamma}$	Branch transition probability
G	Generator matrix
H()	Horizontal runlength
H	Parity check matrix
I{}	Information bit sequence
I	Identity matrix or iterations
K	Constraint length
l	Filter length
$\mathbf{L_c}$	Channel reliability factor
$\mathbf{L_{ek}}$	Soft extrinsic information on symbol k
$\mathbf{L_k}$	Soft information on symbol k including channel information
$\mathbf{L_{uk}}$	A priori soft information on symbol k
$\mathbf{L_S}$	Number of bits in an infomation segment
$\mathbf{m_{ce}}$	Channel estimation metric
$\mathbf{m_{cef}}$	Filtered channel estimation metric
$\mathbf{n_{cwh}}$	Number of high certainty codewords
$\mathbf{n_e}$	Number of estimated bit errors

$\mathbf{n_{fe}}$	Number of actual bit errors
$\mathbf{n_k}$	Noise component of information symbol *k*
$\boldsymbol{\mu}$	Average (mean)
M	Multifold fold parameter or memory of a convolutional code
N, $\mathbf{N_B}$	Information frame size
$\mathbf{N_G}$	Number of segments in an information sub-frame
$\mathbf{N_o}$	Power spectral density of white noise (in Watts/Hz)
$\mathbf{N_S}$	Number of segments in an information frame
π, Π	Interleaver
p	Probability density or parity symbol
P{}	Parity bit sequence
P	Power or probability
p(x|u)	Conditional probability of 'x' given the condition 'u'
$\mathbf{p_k}$	Received parity symbol for the k^{th} information symbol
P	Coefficient matrix
$\mathbf{PW_k}$	Progress codeword for k^{th} information symbol
R	Code rate
$\mathbf{R_{eff}}$	Effective code rate
$\mathbf{R_t}$	Information rate in bits/second
$\mathbf{r_{cwh}}$	High certainty codeword ratio
$\mathbf{r_k}$	Noisy information symbol *k*
R(τ)	Autocorrelation function
$\boldsymbol{\sigma}$	Standard deviation
S()	Square picture element
S(f)	Spectral power density function
$\boldsymbol{\tau}$	Small period of time
T	Time or period
$\mathbf{T_{cw}}$	Total number of crossover codewords
$\mathbf{T_{cwh}}$	Total number of high certainty crossover codewords
U{}	Information symbol sequence
$\mathbf{U_k}$	Received noisy information symbol k
$\mathbf{U`_k}$	Hard value of the decoded information symbol k
V()	Vertical runlength
w	Codeword weight
X{}	Parity symbol sequence
X	Received symbol vector
$\mathbf{x_k}$	Noise free information symbol *k*