University of Alberta

Differentially-Encoded Turbo-Coded Modulation with APP Phase and Timing Estimation

by

Sheryl Lynn Howard ©

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Department of Electrical and Computer Engineering

Edmonton, Alberta
Spring 2007

Your file Votre référence
ISBN: 978-0-494-29687-5
Our file Notre référence
ISBN: 978-0-494-29687-5

Abstract

An easily-implemented serially-concatenated coding system is presented that can operate without channel state information (CSI) via a low-complexity phase estimation technique integrated into the iterative decoding loop. The serially-concatenated code (SCC) is comprised of an outer binary code, separated by a bit interleaver from an inner differential modulation code. When iteratively decoded according to turbo decoding principles, this coding system provides excellent near-capacity performance.

The phase estimation technique utilizes the inner decoder's *a posteriori* probabilities (APP) on the transmitted symbols to form an iteratively-improving phase estimate in each symbol interval, and is termed APP phase estimation. Use of the differential inner code provides rotational invariance to symbol constellation phase slips, as well as interleaver gain due to being a recursive code.

Coherent decoding, with perfect CSI, provides bit error rate (BER) performance 0.6 dB from 8-PSK capacity for large interleaver sizes. Incorporation of APP phase estimation in the iterative decoding process provides near-coherent results for a constant phase offset, without the use of pilot symbols or a training sequence to provide initial CSI; random walk and linear phase models show only fractions of a dB performance loss, compared to perfect CSI, with use of APP phase estimation.

Similarly, timing estimation is incorporated within the iterative decoding loop, and the APP symbol probabilities used to generate iteratively-improving symbol estimates for the classic Gardner timing error detector. This technique is termed APP timing estimation, and achieves near-coherent performance with both a fixed timing offset of $0.4T$, where T is the symbol transmission time, and a uniformly random timing offset distributed over $[-T/2, T/2]$.

A short-cycle-free code-matched interleaver design is also presented that specifically eliminates minimum distance error events in this code. Use of a 6-cycle-free interleaver design lowers the error floor of this coding system by a factor of 50 compared to random interleaving.

Both APP phase and timing estimation were then combined in the iterative decoding loop. Near-coherent performance results were achieved with a phase offset of $\pi/16$ radians and a Gaussian-distributed timing offset.

Acknowledgments

I would like to first and foremost thank my husband, Walt Howard, my parents, Glen and Donna Rattlingourd, and my sisters, Patti Kunkel and Karin Herrmann, for their unflagging support, love and faith in me during the years in which I conducted my doctoral research. Their belief in me sustained me through difficult times, and they shared my joy when the research went well.

Many thanks to the professors on both my candidacy and defense committees for their generous counsel, suggestions, and guidance.

A very special thank-you goes to my advisor and supervisor, Dr. Christian Schlegel. His encouragement, support, mentorship and generous expenditures of his time have been of incalculable value to me during the course of my doctoral degree. Dr. Schlegel has been a superb role model in his example of how to blend the many demands of a professor, teacher, researcher, writer, and mentor.

Thanks go to all my collaborators and friends who contributed to this and other areas of my ongoing research with their interest and support: Drs. Stephen Bates, Bruce Cockburn, Ivan Fair, Vincent Gaudet, Warren Gross, David Haley, Kris Iniewski, Dileepan Joseph, Roland Kempter, Frank Kschishang, Witold Krzymien, John O'Loughlin, Lance Pérez, Byron Schmuland, Dmitry Truhachev and Chris Winstead; Amirhossein Alimohammed, Lovisa Bjorkland, Zhengang Chen, Saeed Fard, Paul Greidanus, Robert Hang, Kees den Hartigh, Sheehan Khan, Lukasz Krzymien, Tyler Brandon Lee, Camille Leroux, Boon Chin Lim, Sarah McEvoy, Sumeeth Nagaraj, Dave Nhan Nguyen, Kerri O'Loughlin, Tony Rapley, Charmaine Ramdass, Ramkrishna Swamy, Saeed Tehrani, Mimi Yiu, Siavash S. Zeinoddin; and all the students of the HCDC and VLSI laboratories in the Electrical and Computer Engineering Department of the University of Alberta.

Very special thanks to my good friends Eliza Detherage, Dr. Cynthia Furse and Dr. Sally McKee, for their counsel, inspiration, support, and reminding me that academics can still have fun!

This thesis is dedicated to the memory of our beloved Ginger, canine companion extraordinaire, who waited patiently, resting at my feet during the night, as I worked late. Her love and loyalty will never be forgotten.

Contents

List of Figures

List of Tables

List of Symbols

a	Transmit symbol
$\mathbf{A}$	Information portion of systematic parity check matrix $\mathbf{H}_{\text{sys}}$
A	*a priori* LLRs in EXIT analysis
$A_{d_i^2}$	Average multiplicity of d_i^2 error events
$A_{d^2_{\min}}$	Average multiplicity of MSED $d^2_{\min}$ error events
α	Exponential decay parameter for channel estimation filter
α_0	Initial exponential decay parameter for channel estimation filter
α_1	Final exponential decay parameter for channel estimation filter
β	Feedback timing offset convergence parameter; also, trellis decoding metric
c	Speed of light
C	Capacity in bits/sec
C_d	Capacity per dimension in bits/symbol
C^*	Capacity for discrete signalling constellation
$CW2$	Two parity codeword bits permuted to last bits of adjacent symbols
$\overline{CW2}$	Two parity codeword bits not permuted to last bits of adjacent symbols
d	Euclidean distance
d_H	Hamming distance
$d_{\min}$	Minimum Hamming distance, minimum distance
$d^2_{\min}$	Minimum squared Euclidean distance (MSED)
D	Delay element; also, # of discretized τ values
$\mathbf{e}$	Decoder error event, bit difference $\mathbf{v}' - \hat{\mathbf{v}}'$
E	Extrinsic LLRs in EXIT analysis
$E[x]$	Expectation of random variable $\mathbf{x}$
$E_T[x]$	Expectation of doubly-truncated Gaussian-distributed r.v. $\mathbf{x}$
E_b	Energy per bit
E_c	Energy per chip

E_s	Energy per symbol
$E_{d_{\min}}$	$d_{\min}$/MSED two-branch error event
E_{dH4}	$d_H = 4$/MSED two-branch error event
E_{par}	$d_{\min}$ error event with > 2 branches;
ϵ	Timing error estimate: output of timing error detector (TED)
$\text{erf}(x)$	Error function of x
f_c	Carrier frequency
Δf	Frequency offset between transmitter and receiver
f_k	Channel estimation filter tap k
$F(\omega)$	Channel estimation filter frequency response
$F(X)$	Gaussian cdf of x at $x = X$; $P(x \leq X)$
$F_T(X)$	Doubly-truncated Gaussian-distributed cdf of x at $x = X$
$g(D)$	Convolutional generator polynomial
$\mathbf{G}$	Generator matrix for block code
$\mathbf{h}$	Channel response
$\hat{\mathbf{h}}$	Instantaneous channel estimate
$\hat{\mathbf{h}}^i$	Instantaneous channel estimate at iteration i
$\hat{\mathbf{h}}_{\text{norm}}$	Normalized instantaneous channel estimate
$\tilde{\mathbf{h}}$	Filtered channel estimates
$\tilde{\mathbf{h}}_{\text{for}}$	Filtered channel estimates, forward pass
$\tilde{\mathbf{h}}_{\text{back}}$	Filtered channel estimates, backward pass
$\mathbf{H}$	Parity check matrix for block code
$\mathbf{H}_{\text{sys}}$	Systematic parity check matrix for block code
$H(D)$	Channel response with delay; partial response channel definition
i	Index for iterations
I	Identity matrix; also, # of interpolation filter taps
I_{conv}	# of decoding iterations until convergence check for cycle-slip detection
$I_{\max}$	Maximum # of decoding iterations

I_{new}	# of additional decoding iterations for cycle-slip detection
I_A	Mutual information $I(x; A)$ between bits x and *a priori* LLRs A
I_{A_i}	Inner decoder *a priori* mutual information
I_{A_o}	Outer decoder *a priori* mutual information
I_E	Mutual information $I(x; E)$ between bits x and extrinsic LLRs E
I_{E_i}	Inner decoder extrinsic mutual information
I_{E_o}	Outer decoder extrinsic mutual information
k	# of information bits; also, index for bits and symbols
$L(\tau)$	Likelihood function with timing offset τ
λ	Log-likelihood ratio (LLR); also, wavelength
λ_a	*a priori* LLR in iterative decoder
λ_{APP}	*a posteriori* LLR output from APP decoder
λ_e	Extrinsic LLR in iterative decoder
$\lambda_{\max}$	Maximum LLR value for decoding
M	M-PSK symbol alphabet; also, # of parity codewords and 8-PSK symbols
m	Index for M-PSK symbols; also, degree of Lagrange polynomial
M_+	$E[\lambda \mid \lambda > 0]$
M_+	$E[\lambda \mid \lambda < 0]$
μ	Mean, expected value of a r.v.
$\mu_{\lvert\lambda_a(v')\rvert}$	Mean *a priori* LLR magnitude into inner APP decoder
μ_T	Mean of doubly-truncated Gaussian-distributed r.v.
$\mathbf{n}$	Zero-mean Gaussian-distributed noise, AWGN
N	Length of codeword $\mathbf{v}$
N_0	AWGN Channel Noise Power Spectral Density
$p_{b,\text{fail}}$	Probability of bit error due to decoding failure
$P_{c \to e}$	Pairwise error probability (PEP) of decoding $\mathbf{e}$ when $\mathbf{c}$ was sent
$p(t)$	Pulse shape in time

$P(\omega)$	Pulse frequency response
$p_{RC}(t)$	Raised-cosine pulse, time-domain response
$P_{RC}(\omega)$	Raised-cosine pulse, frequency-domain response
$P_{RRC}(\omega)$	Root-raised cosine pulse, frequency-domain response
p_a	*a priori* probabilities in iterative decoder
$p_{DTN}(x)$	Doubly-truncated Gaussian pdf
P_e	Probability of error
p_e	Extrinsic probabilities in iterative decoder
$p_N(x)$	Normal or Gaussian pdf
$p(\mathbf{x}\|\mathbf{y})$	*a posteriori* probabilities
$p(\mathbf{y}\|\mathbf{x})$	Channel probabilities
$\mathbf{p_1}, \mathbf{p_2}$	Parity sequences of PCCC
$P(\overline{2\text{adj}})$	Probability two 8-PSK symbols are not adjacent
$P(3\text{adj})$	Probability two or more of three 8-PSK symbols are adjacent
$P(\overline{3\text{adj}})$	Probability two or more of three 8-PSK symbols are not adjacent
$P(CW4)$	Probability between 2 codewords, a $d_H = 4$ 2-branch error exists
$P(\overline{CW4})$	Probability between 2 codewords, no $d_H = 4$ 2-branch error exists
$P(W4)$	Probability interleaver has ≥ 1 $d_H = 4$ two-branch error pattern
$P(\overline{W4})$	Probability interleaver has no $d_H = 4$ two-branch error pattern
Π	Interleaver
$\Delta\varphi(t)$	Time-varying phase offset between transmitter and receiver
$\Delta\varphi$, Φ	Constant phase offset between transmitter and receiver
$Q(x)$	Gaussian Q-function
$r(t)$	Received noisy carrier-modulated pulse-shaped signal
R	System rate
$s(t)$	Carrier-modulated pulse-shaped transmitted signal
σ	Standard deviation of Gaussian-distributed random variable
σ_T	Standard deviation of doubly-truncated Gaussian-distributed r.v.

σ_n^2	AWGN channel noise variance
S	Signal power, also spreading amount for spread interleaver
S/N	Signal to noise power ratio
$S(I)$	8-PSK symbols that bits of codeword I interleave to
$S(I)_2$	8-PSK symbols that bits of $V(I)_2$ not connected to $X(I)_1$ interleave to
t	Time
T	Symbol timing
T_s	Sampling time
Δt, τ	Timing offset between transmitter and receiver clocks
τ_D	Discretized values of timing offset τ for maximal search
$\hat{\tau}$	Estimate of timing offset τ
$\hat{\tau}^i$	Estimate of timing offset τ in iteration i
$\hat{\tau}_{\mathrm{DD}}$	Data-directed timing offset estimate
$\hat{\tau}_{\mathrm{DA}}$	Data-aided timing offset estimate
$\hat{\tau}_{\mathrm{NDA}}$	Non-data-aided timing offset estimate
θ	Zero-mean Gaussian-distributed random phase offset
$\mathbf{u}$	Information or data bit sequence
$\hat{\mathbf{u}}$	Estimated information or data bit sequence
$\mathbf{u}'$	Interleaved information or data bit sequence for PCCC
$u(D)$	Convolutional information polynomial
$\mathbf{v}$	Codeword; also, specifically codeword from outer code of SCC
$\mathbf{v}'$	Interleaved $\mathbf{v}$ for SCC
$v(D)$	Convolutional codeword polynomial
v	Velocity
$V(I)$	Set of bits in (3,2,2) parity codeword I
$V(I)_2$	Set of (3,2,2) codewords that $X(I)_1$ deinterleave to
$V(I)_3$	Set of (3,2,2) codewords that $X(I)_2$ deinterleave to
ν	Constraint length

W	Signal or symbol pulse bandwidth
$W2$	Random interleaver permutes 2 cw bits to last bits of adjacent symbols
$\overline{W2}$	Random interleaver does not permute 2 cw bits to adjacent symbol last bits
ω	Frequency in radians
$\mathbf{x}$	Transmitted symbol
$\hat{\mathbf{x}}$	Estimated transmitted symbol
$\mathbf{x}_0$	Known training sequence of transmitted symbols
$\tilde{\mathbf{x}}$	Soft symbol estimate
$\mathcal{X}$	Symbol alphabet of x
$X(I)_1$	Set of 8-PSK symbols directly adjacent to $S(I)$
$X(I)_2$	Set of 8-PSK symbols directly adjacent to $S(I)_2$
$\mathbf{y}$	Received noisy transmitted symbol
$\mathbf{y}^i$	Interpolated received sequence at iteration i

List of Acronyms

APP	*a posteriori* probability
ASK	Amplitude-shift keying
AWGN	Additive white Gaussian noise
AZCW	All-zeros codeword
BPSK	Binary phase-shift keying
BCH codes	Bose-Chaudhuri-Hocquenghem codes
BCJR algorithm	Bahl-Cocke-Jelinik-Raviv algorithm
BER	Bit error rate
BICM-ID	Bit-interleaved coded modulation with iterative decoding
CCITT	Comité Consultatif International Téléphonique et Télégraphique
cdf	Cumulative distribution function
CDMA	Code-division multiple-access
CSI	Channel state information
DA	Data-aided
df	Decoding failure
DD	Data-directed
DRP interleaver	Dithered relative prime interleaver
DTTL	Data-transition tracking loop
DVB-S2	Digital video broadcasting satellite link standard
ECC	Error-correcting codes or error-control codes or coding
EM	Expectation-maximization
ESA	European Space Agency
EXIT	Extrinsic information transfer
FSM	Finite state machine
GTED	Gardner timing error detector
ISI	Inter-symbol interference

ITU	International Telecommunications Union
LDPC	Low-density parity-check code
LLR	Log-likelihood ratio
LO	Local oscillator
MAP	Maximum *a posteriori*
ML	Maximum likelihood
M&M TED	Mueller & Müller timing error detector
MPSK	M-ary phase-shift keying
MS	Mode separation
MSE	Mean squared error
MMSE	Minimum mean squared error
MSED	Minimum squared Euclidean distance
MSEW	Maximal squared Euclidean weight
NDA	Non-data-aided
NSD	Non-coherent sequence detection
OOK	On-off keying
PCCC	Parallel concatenated convolutional code
PCTCM	Parallel concatenated trellis-coded modulation
PEP	Pairwise error probability
PLL	Phase-locked loop
pdf	Probability density function
PML	Pseudo-ML
ppm	Parts per million
PR4	Partial response channel type
PSAM	Pilot symbol assisted modulation
PSP	Per-survivor processing
QAM	Quadrature amplitude modulation
QPSK	Quadrature phase-shift keying

RP interleaver	Relative prime interleaver
RSC	Recursive systematic convolutional code
r.v.	Random variable
SCC	Serially-concatenated code
SCCC	Serially-concatenated convolutional code
SCTCM	Serially-concatenated trellis-coded modulation
SED	Squared Euclidean distance
SISO	Soft-input soft-output
SNR	Signal-to-noise ratio
SOVA	Soft-output Viterbi algorithm
TCM	Trellis-coded modulation
TED	Timing error detector
TEE	Turbo-embedded estimation
TTCM	Turbo trellis-coded modulation
VA	Viterbi algorithm
VCO	Voltage-controlled oscillator
XOR	Exclusive-OR
8-PSK	8-ary phase-shift keying

Chapter 1

Introduction

With the advent of turbo codes [1], [2] and iterative decoding of serially concatenated codes [3] (SCCs), the push for near-capacity performance of error control codes has become a reality. Use of these near-capacity codes with iterative decoding now allows receiver operation in very high noise/low signal-to-noise ratio (SNR) environments.

This thesis approaches the construction of an SCC for higher order modulations from a code design perspective. Use of an inner differential M-PSK modulation code is shown to lead naturally to a decoder which functions without any prior channel phase information in the form of pilots or a training sequence, providing performance close to that of a receiver with complete phase knowledge, *i.e.*, coherent decoding, with several different phase models.

Some background on basic digital communications systems and the need for phase and timing synchronization between transmitter and receiver will be presented in Chapter 2. Error-control coding and Shannon's capacity limit are discussed in Chapter 3.

1.1 Motivation

One of the primary goals of the research presented in this thesis was to design an error-control-coded system which would provide near-capacity performance along with moderately high throughput with an easily-implemented code. These goals can be achieved by appropriate choice of an iteratively-decoded concatenated code, using higher-order modulation, and easily-implemented component codes. An analysis of some of the design parameters for a near-capacity code design with good error floor performance as well are presented in Chapter 6.

Near-capacity performance means the coded system will be operating at very low SNR, compared to an uncoded system. Additionally, the higher-order modulation, while increasing throughput, makes the system more sensitive to noise. Synchronization algorithms which work well at SNRs for which the uncoded system provides good performance face significant challenges when operating at the much lower SNRs where near-capacity-achieving codes can function. If timing and phase estimation are performed before decoding according to the classic model of a communications system, they may not achieve synchronization; this lack of synchronization may then cause decoder failure also.

Incorporating timing and phase estimation algorithms within the iterative decoding loop allows knowledge of the code structure to be utilized in the form of the probabilistic information (called soft information) available from the decoder. This soft information can be used as soft symbol decisions for the estimation algorithms, which then can update the channel probabilities in the next iteration of decoding.

Some recent estimation methods incorporate phase or timing estimation either in the decoder structure or within the iterative decoding loop. These methods, as well as the method presented in this thesis, termed **APP channel and timing estimation** because the *a posteriori* probabilities in Equation 2.4 available from the APP or MAP decoder are used to form soft symbol estimates used in the channel and timing estimation algorithms, are presented in Chapters 9, 10 and 11.

1.2 Thesis Contributions

This thesis presents an error-control-coded communications system with phase and timing estimation incorporated into the iterative decoding process. Many facets of a communications system are explored: error-control code design, modulation, bit mapping matched to maximize overall code performance, interleaver design matched to the component codes for good error floor performance, and both phase estimation and timing estimation utilizing the inner decoder's soft information output.

1.2.1 Code Design

A serially-concatenated code design composed of simple, easily-implemented component codes which provides near-capacity performance results with a throughput of 2 bits/symbol is presented and analyzed. An EXIT analysis approach [4], [5] is used to find matched outer codes that work optimally with the inner differential modulation code. The (3,2,2) parity check code is such a code, providing turbo cliff results only 0.6 dB away [6], [7] from the 8-PSK capacity at a system rate of 2 bits/symbol [8], [9].

1.2.2 Modulation and Bit Mapping

Differential 8-PSK modulation is used as the inner code to provide higher throughput, rotational invariance to symbol constellation phase slips, which is advantageous for channel estimation, and a good match to the outer parity code for near-capacity performance. The effect of bit mappings on error-floor performance is explored by examining minimum distance error events for this code design, with the goal of maximizing overall code performance with respect to both waterfall and error floor performance.

1.2.3 Interleaver Design

A code-matched interleaver specifically designed to eliminate minimum distance error events for the concatenated code is presented. The design is developed by viewing the code/interleaver interface in factor graph representation. Minimum distance error events are shown as short-length cycles. An interleaver designed without these specific types of short cycles significantly lowers the concatenated system's error floor, by nearly two orders of magnitude.

1.2.4 APP Phase Estimation

A phase estimation technique that utilizes the *a posteriori* probabilities available from the inner APP decoder to generate soft symbol estimates is presented. The soft symbol estimates are not forced to hard decisions, but instead can incorporate an existing phase offset. Instantaneous channel estimates are calculated from the soft symbol estimates and received symbols, and then filtered to provide more stable channel estimates. The channel estimates incorporate the phase offset, and are used to generate better channel probabilities for the next decoding iteration.

The inner differential APP decoder relies on the ability of the APP decoder for the differential inner modulation code to provide useful soft information on the information or data symbols without any prior phase knowledge, even in the presence of significant time-varying channel phase offset, which is confirmed via EXIT analysis [4], [5]. Through iterations, decoding performance close to the complete phase knowledge scenario is achieved with integrated APP channel phase estimation.

This iteratively-improving phase estimation technique has relatively low complexity compared with the APP decoding operation. The phase estimation is not directly incorporated into the APP decoding trellis, which would significantly expand the trellis complexity, but is outside the decoder, yet within the iterative decoding loop. Neither external phase estimation, such as a training sequence preamble or non-data-aided (NDA) phase estimation, nor differential demodulation are needed to begin the decoding and estimation process.

Performance for this concatenated coding system with incorporated APP phase estimation is presented for three different phase models: a constant phase offset, a Gaussian random walk time-varying phase, and a constant frequency offset/linearly-increasing phase offset. The constant phase offset model shows performance equivalent to perfect phase synchronization with phase offsets $\leq \pi/16$ rads; performance degrades as the offset approaches $\pi/8$ rads. The random walk and linear phase offset show fractions of a dB loss in waterfall performance, with error floor performance equivalent to that with perfect timing, except for higher linear phase offset which exhibits a slightly raised error floor.

1.2.5 APP Timing Estimation

In a similar manner, timing estimation is incorporated into the iterative decoding loop. The same method of obtaining soft symbol estimates from the inner APP decoder used in APP channel estimation is used here, and thus termed APP timing estimation. A classic decision-directed timing estimation algorithm is used, with the soft symbol estimates used in place of hard symbol decisions.

1.2.6 Combined APP Phase and Timing Estimation

Both APP phase and timing estimation are incorporated within the iterative decoding loop, utilizing the soft symbol estimates obtained from the APP probabilities of the inner differential 8-PSK decoder. In this way, both synchronization and decoding are combined into one integrated block, each enhancing the performance of the other section.

1.3 Thesis Outline

The thesis is organized in the following manner. First, a brief introduction to communications systems is presented in Chapter 2 as foundational material. Chapter 3 discusses basics of error-control coding and channel capacity. Capacity-approaching iteratively-decoded codes such as turbo codes are described in Chapter 4. Chapter 5 describes the serially concatenated binary error control code/differential 8-PSK system presented in this thesis, examining the encoder, simulated channel, and decoder. Analysis techniques used to improve the system are discussed in Chapter 6; distance spectrum analysis in Section 6.1 allows improvement of the error floor through a new mapping, while EXIT analysis in Section 6.2 predicts turbo cliff behavior and completes the error analysis. Performance results for several 8-PSK mappings, interleaver sizes and two different outer rate 2/3 binary codes are presented for the differentially-encoded turbo-coded modulation system in Chapter 7. A code-matched interleaver design to further improve the system's error floor is developed in Chapter 8. This interleaver specifically eliminates low-weight error events in the system by viewing them as short cycles and avoiding these short cycles in the interleaver design process.

Chapter 9 presents a method of obtaining channel estimates from the inner APP decoder when decoding without CSI, termed **APP channel estimation**. Simulation results for several phase models are also presented. Timing estimation is considered in Chapter 10, and **APP timing estimation** is presented. In a similar manner to APP channel estimation, APP timing estimation utilizes the APP probabilities available from the inner APP decoder for estimation purposes, generating a new timing estimation with each decoding iteration. Simulation results for APP timing estimation are presented for a constant timing offset. Finally, in Chapter 11, both APP phase and timing estimation are integrated into the iterative decoding loop, to form a combined decoder/synchronizer. Conclusions are discussed in Chapter 12.

Chapter 2

Digital Communications Systems

2.1 A Basic Communications System

A basic digital communications system consists of a transmitter, a receiver and the channel, which is the environment existing between transmitter and receiver. Digital data **u** are input to the transmitter, converted to a form better suited to transmission (discussed in Section 2.2), and sent across the channel to the receiver. The receiver then makes a decision on what the original digital data **u** were, and outputs the decision as $\hat{\mathbf{u}}$. A block diagram of the basic communications system is shown in Figure 2.1. Many details are hidden within the transmitter and receiver blocks, and not shown in this basic communications system view, but are considered in the following sections.

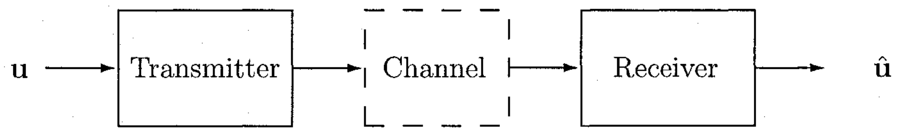


Figure 2.1: A basic digital communications system.

Two examples of communications systems are a desktop computer sending a file to a printer, and a cell phone sending a conversation to the nearest base station. The computer/printer system is a wired communication system; the data travels from computer to printer via a cable. The cell phone/base station system is a wireless communication system, where the cell phone data is transmitted from the phone antenna through the air to wherever the closest base station is located. The cell phone/base station channel sees data signal loss due to attenuation of the transmitted signal as it travels through the air, and possible scattering losses from obstacles such as trees, buildings, cars, hills or even rain in the path of the signal, as well as interference from other cell-phone users in the area. The computer/printer channel sees loss due to attenuation in the cable and loss at the connectors; this loss can also be significant. Channels can vary significantly in the amount of signal loss incurred by the channel, depending on the system.

The properties of the channel are described in probabilistic terms; a channel model very commonly used is the additive white Gaussian noise (AWGN) channel, which quite accurately models thermal noise at the receiver as a zero-mean Gaussian process added to the transmitted data. "White" refers to the noise samples having a flat frequency spectrum over the bandwidth of interest, which is the signal bandwidth. Channel impairment is described in this thesis as **noise**.

If the transmitter sends data $\mathbf{x}$ across an AWGN channel, the receiver obtains

$$\mathbf{y} = \mathbf{x} + \mathbf{n}, \quad n : \mathcal{N} : (0, \sigma_n^2 = N_0/2) \tag{2.1}$$

due to AWGN with noise variance $\sigma_n^2 = N_0/2$ per dimension and $E[n_i n_j] = 0$ if $i \neq j$. No channel fading or intersymbol interference (ISI) are considered here.

2.2 Data Modulation

Binary data typically are in the form of ones and zeros. The binary data may be sent directly, where each transmitted bit $x \in [0, 1]$; this is known as on-off keying (OOK) or amplitude-shift keying (ASK). However, this method is not very practical for packet messaging systems, where the receiver may not know when a packet will be arriving; in that case, the receiver cannot tell the difference between an all-zeros codeword (AZCW) or no packet reception, as both cases consist only of noise without any signal magnitude.

A better binary modulation choice for packet messaging is to send $x \in [-1, 1]$, with a 0 bit mapping to -1 (for example). This is termed antipodal signalling or binary phase-shift keying (BPSK). Phase-shift keying refers to the fact that the data is contained in the signal phase, so that 0^o phase means a 1 and 180^o phase means -1. The receiver can more easily differentiate a received noisy $\mathbf{y} = \mathbf{x} + \mathbf{n}$ from receiving simply noise. Additionally, because the two symbols are further apart, by a distance 2 in BPSK rather than 1 for OOK, the receiver can detect whether a 1 or -1 was sent in the presence of greater noise than for OOK. Better performance in noise is achieved at the cost of higher signal energy expenditure.

These binary signalling schemes convey one bit of information per transmitted symbol. However, in these days of massive information transfer, high throughput, or the amount of data transmitted per time, is very important. Modulation schemes have been developed which increase the throughput above one bit per symbol. In other words, each symbol contains more than one bit.

M-ary phase shift keying (M-PSK) is a modulation scheme that converts $\log_2(M)$ bits into one of M possible symbols, for a throughput of $\log_2(M)$ bits/symbol. The information is again contained in the phase, with all symbols having equal magnitude and being equally spaced points on a circle. The M possible symbols of M-PSK modulation are denoted by $x \in \exp(j2\pi m/M), \forall m = 0, \ldots, M-1$, where $j = \sqrt{-1}$.

An 8-PSK constellation is shown in Figure 2.2. Each signal point is an angular distance $\pi/4 = 45^o$ apart, and is represented by three data bits. The decimal equivalent of the three bits gives the symbol label. Natural mapping of the bits, which maps the three-bit combinations in a consecutive, numerically-increasing manner counter-clockwise from 0^o to 315^o, is used in the figure.

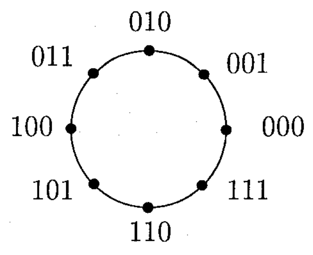


Figure 2.2: 8-PSK symbol constellation with natural bit mapping.

2.3 Carrier Frequency and Pulse-Shaping

The data symbols must be transmitted to the receiver in some fashion, either along wires or through the air. The symbols are used to modulate a periodic waveform which can propagate down the wires or from a transmitting antenna through the air to a receiving antenna. Because this waveform carries the modulated data symbols, it is known as a carrier wave. Typically the carrier wave will be a radio-frequency (RF) sinusoid; high frequency reduces the required antenna size for a $\lambda/2$ antenna to a reasonable size.

Viewing the data symbols as either an impulse or rectangular pulse which modifies the carrier wave at intervals of the symbol timing T is not practical, because both impulses and rectangular pulses in the time domain have infinite extent in the frequency domain, or infinite bandwidth. This topic is discussed further in Chapter 10. The pulse train formed by the transmitted symbols should be bandlimited. Thus the symbols need to be first multiplied by a pulse $p(t)$, or equivalently filtered through a pulse-shaping filter $P(\omega)$. Then the symbol pulses are multiplied by the carrier wave of frequency f_c. This is known as upconversion because $f_c \gg 1/T$, where $1/T$ is the symbol transmission rate. The symbol pulse bandwidth W should be within $1/T$ and thus is much less than the carrier frequency f_c.

The pulse-shaped symbols, before multiplying with the carrier, have frequency response only within $W > f > -W$, and is known as a **baseband** signal. All operations prior to modulation with the carrier wave occur at baseband frequencies. The carrier-modulated signal $s(t)$ is known as a **bandpass** signal, because its frequency response is zero outside of the range $f_c + W > f > f_c - W$ and $-f_c + W > f > -f_c - W$. Various pulse shapes are discussed in Chapter 10.

2.4 Receiver

The end function of the receiver in a digital communications system is to generate an estimate $\hat{\mathbf{u}}$ of the original digital data $\mathbf{u}$. To do this, the receiver must undo the operations which the transmitter performed on the data in order to be able to send the data across the channel and for the receiver to obtain the data. The following elements of a digital communications receiver are presented in a typical order; however, these operations may be performed in different order depending on the specific system.

First, the receiver moves the received signal $r(t)$ from bandpass frequencies down to baseband by multiplying it with a replica of the carrier wave at f_c. This multiplication of a sinusoid with itself generates a baseband copy of the signal as well as duplicates centered at $\pm 2f_c$. A low-pass filter removes these duplicates.

Filtering this signal through a pulse-matched filter whose frequency response is $P(\omega)$ both low-pass filters the carrier-multiplied received signal and correlates the pulse $p(t)$ with itself. This correlation is maximized (assuming correct timing) when a transmitted symbol is actually present; if only Gaussian noise is present, the output of the matched filter will be simply zero-mean noise. The pulse-matched filter is followed by a sampler, which takes discrete samples of the filtered signal spaced at intervals of T (or less, for applications which require oversampling). If the receiver clock is synchronized with the transmitter clock, so that the receiver samples each pulse at the peak of the pulse, the symbol samples will have maximum magnitude.

The receiver then must estimate what the original information sequence $\mathbf{u}$ was, given these discrete samples. This process is known as symbol **detection** or **demodulation**. We assume the peak symbol samples are used for demodulation.

2.5 Detection/Demodulation

In demodulation, the receiver chooses the most likely data symbols $\mathbf{x}$ sent to it, based on the received samples $\mathbf{y}$ and knowledge of the symbol alphabet $\mathcal{X}$. If the effect of the channel is known or can be estimated, then the channel probabilities $p(\mathbf{y}|\mathbf{x})$ can be determined. For the AWGN channel with zero-mean Gaussian noise of variance $\sigma_n^2 = N_0/2$ per dimension, the channel probabilities are found as

$$p(\mathbf{y}|\mathbf{x}) = p_n(\mathbf{y}-\mathbf{x}) = \frac{e^{-|y-x|^2/N_0}}{\sqrt{\pi N_0}}. \tag{2.2}$$

Maximum likelihood (ML) decoding chooses the most likely sequence $\mathbf{x}$ according to the ML decision

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} p(\mathbf{y}|\mathbf{x}), \tag{2.3}$$

where $p(\mathbf{y}|\mathbf{x})$ is the conditional probability of the received sequence $\mathbf{y}$ given the transmitted sequence $\mathbf{x}$. ML decoding optimizes the sequence probability.

Maximum *a posteriori* (MAP) decoding chooses the most likely bits or symbols x_k according to the MAP criterion

$$\hat{x_k} = \arg\max_{x_k} p(x_k|\mathbf{y}), \tag{2.4}$$

where $p(x_k|\mathbf{y})$ is the *a posteriori* conditional probability of symbol x_k given the received sequence $\mathbf{y}$. MAP decoding optimizes the bit or symbol probability, rather than the sequence probability. As shown below, if the symbols have equal probability, MAP decoding reduces to ML decoding. Without a method of providing *a priori* information on the bit or symbol probabilities, such as in iterative decoding (discussed further in Chapter 4), MAP decoding provides the same results as ML decoding, with increased decoding complexity.

Using Bayes' theorem that $p(x,y) = p(x|y)p(y)$, and the fact that $p(\mathbf{y})$ is constant for all x and thus has no effect on the maximization,

$$\hat{x_k} = \arg\max_{x_k} p(\mathbf{y}|x_k)p(x_k). \tag{2.5}$$

If $p(x_k)$ is assumed equally likely for all x_k, then it too has no effect on the maximization, and the MAP decision becomes

$$\hat{x_k} = \arg\max_{x_k} p(\mathbf{y}|x_k). \tag{2.6}$$

Assuming independence between symbols x and also between noisy received symbols y, we note that only y_k depends on x_k and so $p(\mathbf{y}|\mathbf{x}) = \prod_k p(y_k|x_k)$. Thus MAP decoding with equal probabilities $p(x_k)$ becomes

$$\hat{\mathbf{x}} = \prod_k \arg\max_{x_k} p(y_k|x_k), \tag{2.7}$$

reducing to the maximum likelihood (ML) decision of Equation 2.3.

Assuming no error-control coding, and equally-likely symbols x, ML detection implements Equation 2.3 by choosing the estimated symbol sequence $\hat{\mathbf{x}}$ which maximizes Equation 2.2, either on a symbol-by-symbol basis if there is no coding, comparing each possible value of x, or by use of an ML decoding algorithm such as the Viterbi decoding algorithm [10]. Then the estimated information sequence $\hat{\mathbf{u}}$ is found, knowing the mapping of bits u used for each symbol x.

The use of error-control coding to improve system performance is discussed in Chapter 3. If error-control coding or error-correcting coding (ECC) is used, then knowledge of the code structure is used in the implementation of the decoding algorithm.

2.6 Synchronization

Analysis of communications systems, and of error-control coding techniques, often assumes synchronization between the transmitter and receiver. Synchronization typically has two phases, acquisition and tracking. Acquisition is more difficult than tracking, as it requires the acquisition of an unknown parameter. Acquisition may be achieved in a set-up period before the actual data is transmitted, through use of a known preamble of data. Tracking assumes that initial acquisition of the unknown parameter has been achieved, but that parameter is slowly time-varying, so continuous estimation of the changing parameter is required to maintain synchronization. The use of known pilot symbols spaced at intervals throughout the data can aid in tracking the unknown parameter.

There are three important types of synchronization in digital communications systems:

1. Carrier Synchronization or Phase Synchronization;
2. Timing or Symbol Synchronization;
   and
3. Frame Synchronization.

For packet transmission, where the receiver does not know when a packet is going to arrive, synchronization is done in reverse order, as

1. Frame Synchronization;
2. Timing or Symbol Synchronization;
3. Carrier Synchronization or Phase Synchronization.

The receiver must first decide whether a packet has arrived, which involves synchronizing with the received frame. Usually this achieves synchronization to within one symbol interval, and finer estimation of the timing is then achieved through timing synchronization.

2.6.1 Carrier or Phase Synchronization

The carrier wave generated at the receiver should be synchronized with the transmitter carrier wave; both should be at the same frequency f_c and matched in time. A frequency offset Δf between transmitter and receiver carrier waves generates a linearly time-varying phase offset $\Delta\varphi(t) = \Delta f t$. A time offset Δt between the peaks of transmitter and receiver sinusoids results in a constant phase offset $\Delta\varphi = f_c \Delta t$.

In practice, the voltage-controlled oscillators (VCOs) generating the carrier frequency will suffer some drift in carrier frequency, so that both transmitter and receiver may generate carrier waves with different offsets from the carrier frequencies.

Various carrier and phase estimation algorithms exist to adjust the receiver carrier frequency to match that of the transmitter, and to compensate for any existing phase offset. Phase estimation is discussed further in Chapter 9.

2.6.2 Timing or Symbol Synchronization

Timing synchronization between transmitter and receiver refers to the synchronization of the sampling time at the receiver with the peak pulse timing at the transmitter. Both transmitter and receiver have a clock supplying their timing. Even if both clocks run at the same frequency, so both are sampling at the symbol timing T, there may be a timing offset between transmitter and receiver clocks, so that at transmitter time T, the receiver has time $T + \tau$.

If the receiver sampling time is offset from the transmitter pulse or symbol timing, the receiver samples will be taken when the pulse magnitude is not at a maximum. Worse, because the pulses are typically not time-limited to their own symbol period (discussed further in Chapter 10), other pulses will interfere with the pulse of interest. This interference is known as **intersymbol interference** or ISI, and can be quite severe, depending on the timing offset. Estimation of the timing offset, and correcting for the timing offset in existing samples, is discussed in Chapter 10.

This thesis considers a possible timing offset τ between transmitter and receiver. The clocks may also be offset in frequency, resulting in a time-varying τ, and the possibility of some symbols not being sampled at all. This asynchronicity between transmitter and receiver clocks is not considered in this thesis. The timing offset τ is considered to be constant over a frame or packet.

2.6.3 Frame Synchronization

Data in communications systems is often sent in frames or packets of specific size. If the receiver knows what size packet to expect, but not when, it must decide at what point the frame begins and ends. Otherwise, the receiver will demodulate some noise as symbols at the beginning of the frame and miss detecting other symbols at the end of the frame, or vice versa.

Frame synchronization becomes critical in applications where the entire packet of data must be received in order to be useful. One of these applications is when error-control coding is used at the transmitter, with block codes. These codes are a specific size and are decoded based on knowledge of either all the possible codewords or of the code structure. Block codes and error-control codes in general are discussed further in Chapter 3.

This thesis does not incorporate frame synchronization into the iterative decoding loop. However, frame synchronization affects the quality of initial timing synchronization and sampling available to the iterative decoder. Frame synchronization is considered in greater detail in Section 11.4, where the effect of the initial timing offset distribution is examined.

2.7 Digital Communications System with Synchronization

Typically, a digital communications system incorporates synchronization before demodulation, or before decoding if an error-control code was used at the transmitter. Timing and phase estimation may be combined or separate, depending on the application and estimation algorithms used.

A block diagram of a typical communications system including the blocks discussed above is shown in Figure 2.3, with phase and timing synchronization incorporated as a single synchronization block labeled 'Sync'. Frame synchronization is not shown in the block diagram. Error-control coding, discussed in Chapter 3, is included, with an encoder at the transmitter and the corresponding decoder at the receiver. A decoder typically implements either ML or MAP decoding, or a sub-optimal version thereof, but incorporates knowledge of the code structure instead of simple demodulation based on the channel noise probabilities.

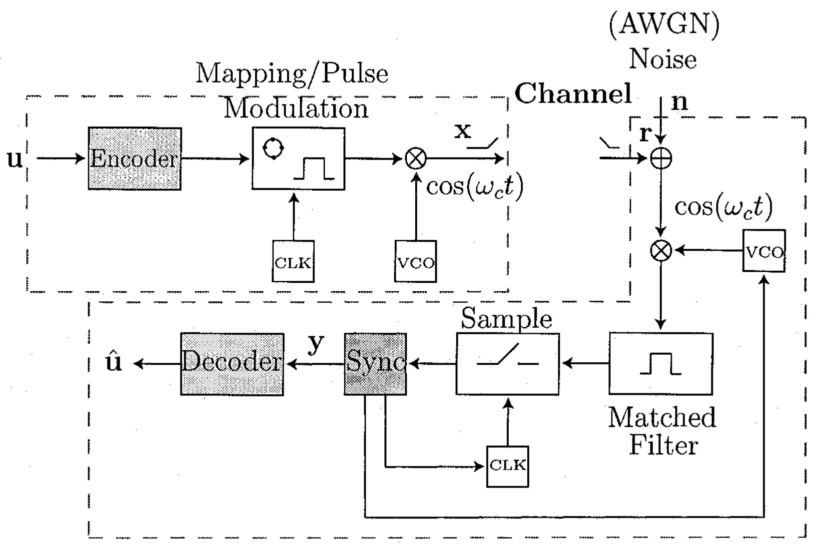


Figure 2.3: Digital communications system, including synchronization and error-control coding and decoding.

2.8 System Performance

A common measure of a communications system's performance is its bit error rate (BER). The average bit error rate is usually based on the original binary data $\mathbf{u}$ and can be measured as $E[\text{BER}]=\sum_{i=1}^{k*N} |u_i - \hat{u}_i|/(k * N)$ for k information bits per frame and N total frames transmitted.

The BER depends upon the channel model; for an AWGN channel, specifying the noise power N_0 or variance $N_0/2$ per dimension characterizes the channel. Typical system performance plots display BER vs signal-to-noise ratio (SNR). The SNR has different definitions, but the most common ones in use for communications systems are in terms of signal energy per bit E_b or signal energy per symbol E_s:

$$\text{SNR} = 10\log_{10}\left(\frac{E_b}{N_0}\right) \quad \text{or} \quad 10\log_{10}\left(\frac{E_s}{N_0}\right) \quad \text{in dB,} \tag{2.8}$$

For equal-energy symbol constellations such as M-PSK modulation, E_s can be normalized to 1 without loss of generality. The energy per bit E_b is found from the energy per symbol E_s as

$$E_b = E_s/R, \tag{2.9}$$

where R is the system rate in terms of input bits per symbol. For 8-PSK modulation, $R = 3$ bits/symbol. For error-control coding, where extra bits are added with the coding to provide redundancy, the rate R can be less than 1 when BPSK modulation is used; for example, a rate 1/2 code produces 2 output bits for every input bit.

The purpose of defining SNR in terms of E_b rather than E_s is to incorporate the effect of system rate R into the performance. If equal-energy modulations such as QPSK and 8-PSK are compared directly using E_s/N_0, they both have the same symbol energy E_s but different bit energies E_b due to differing rates. Using Equation 2.8 for E_b/N_0 instead of E_s/N_0 as the SNR definition to compare performance for both modulations ensures that for a given SNR, both have the same bit energy. Comparison on the basis of SNR=E_b/N_0 seems a fairer measure of performance than SNR=E_s/N_0, as it accounts for the relative energy loss for lower-rate codes.

System error-rate performance in this thesis is presented in terms of BER vs SNR, with SNR defined according to Equation 2.8 in terms of E_b/N_0.

Chapter 3

Error Control Coding and the Drive Towards Capacity

3.1 Introduction

The field of error-control coding developed as a method to extract reliable data from an unreliable, noisy received signal. The receiver chooses the most likely data symbol or sequence sent to it, based on the received signal. However, if the received signal is impaired with too much noise, significant errors will result from the receiver choosing many incorrect bits in the data sequence.

One obvious response to the noise problem is simply to increase the transmitted signal strength, thus increasing the signal-to-noise ratio (SNR) enough for the receiver to make reliable decisions on the data sent. Increasing transmitter power is often not practical, for example, in deep-space communications.

Instead, the philosophy of error-control coding is to improve the receiver's ability to choose the correct data even in the presence of significant noise, that is, at low SNR. Error control coding achieves this by encoding the original digital data sequence, often termed the **information sequence**. The encoded sequence is transmitted and the channel adds some form of noise. At the receiver, the noisy encoded sequence is decoded, usually by

ML or MAP decoding, incorporating knowledge of the encoder or codewords.

The encoding process produces an encoded sequence or codeword $\mathbf{x}$ which typically has extra bits added to the information sequence $\mathbf{u}$ and thus is longer than $\mathbf{u}$. These extra bits, known as **parity bits**, depend in some way on the information bits and thus provide redundancy that can be used to advantage when transmitted over a noisy channel.

A simple example is the repetition code, which repeats the information bits $\mathbf{u}$ a fixed number of times. The (3,1) repetition code has codeword length 3 and information sequence length 1. Any long information sequence to be encoded is considered one bit at a time, and each individual bit is repeated three times before moving to the next information bit. Without encoding, if a single received information bit is noisy enough to be decoded incorrectly, that bit will be in error. However, the (3,1) repetition code must receive two bits of each three-bit codeword in error for the codeword to be decoded correctly. If only one bit is in error, the decoder will correct it.

This is shown in the following example: Suppose the codeword 000 is transmitted, but 100 is received. The (3,1) repetition code only has two possible codewords, 000 and 111, and 100 is not one of those codewords. The decoder will choose 000 as the most likely codeword because only one bit error is required, compared with 111 which requires two bit errors. Two bit errors are less likely to occur than one bit error. In other words, $p(\mathbf{x} = 000|\mathbf{y} = 100) > p(\mathbf{x} = 111|\mathbf{y} = 100)$ and assuming $p(x = 0) = p(x = 1) = 1/2$, ML decoding chooses $\hat{\mathbf{x}} = 000$ as the most likely codeword. Thus the decoder will correct a codeword with one bit error, and only make an error if two bit errors occur.

The extra redundancy supplied by an error control code increases the probability of decoding the information sequence of bits correctly. The decoder provides an estimate or decision of the most likely information sequence corresponding to the most likely codewords, based on the received noisy sequence and the decoder's knowledge of the code and of the channel noise properties.

Error-control coding lowers the decoded error rate at the receiver. This lowered error rate is achieved at the cost of greater decoding complexity. The tradeoff in error-control coding is reduced transmission power for increased decoder complexity. Digital communications applications today typically demand very powerful error control coding and decoding algorithms to achieve nearly error-free performance, and the extremely high density and low cost of silicon technology has made their implementation possible.

3.2 Capacity

There is a fundamental limit to the rate at which information can be reliably transmitted over a channel, known as the **capacity** of the channel. The theory of this limit was developed in 1948 by Shannon [11] in his famous Capacity Equation:

$$C = W \log_2 (1 + S/N) \text{ [bits/s]}. \tag{3.1}$$

C is the capacity, or maximum rate in bits/s, below which there exists a system with bandwidth W and signal-to-noise power ratio S/N can transmit with arbitrarily low bit error rate over an AWGN channel. Transmitting at a rate above capacity results in an irreducible bit error rate.

Shannon's capacity theorem proved that for transmission at a rate below capacity, there exists a code which could be decoded with arbitrarily low error probability, *i.e.*, always decoded correctly. Shannon's proof did not, however, describe such a code.

Capacity may be expressed either in terms of rate or, for a given rate, the SNR that is the minimum SNR at which near-zero bit error rate is possible, according to Equation 3.1. As Equation 3.1 is dependent on the system bandwidth W, a quantity that varies depending on application but is independent of code choice, Shannon's capacity theorem may be converted into a bandwidth-independent version [12] with units of bits/dimension or bits/symbol:

$$C_d = \frac{1}{2} \log_2 \left(1 + 2\frac{RE_b}{N_0}\right) \text{ [bits/symbol]}, \tag{3.2}$$

where R is the overall system or code rate in bits/symbol. Both Equation 3.1 and 3.2

assume a real-valued infinite symbol alphabet with Gaussian distribution is used for a signalling scheme, which provides the best results, in terms of highest achievable rate or minimum required SNR. In practice, an infinite symbol alphabet is not currently possible, and a feasible signalling scheme is constrained to a finite constellation, such as BPSK, QPSK, 8-PSK, 256-QAM, etc. A constrained symbol constellation reduces the achievable capacity. A capacity formulation for discrete symbol constellations over the AWGN channel evaluates capacity at the M constellation points or symbols in [9] as

$$C^* = H(a) - H(a|y) = \log_2(M) - \frac{1}{M}\sum_{k=0}^{M-1} E\left[\log_2 \sum_{i=0}^{M-1} \exp\left\{-\frac{|a_k + n - a_i|^2 - |n|^2}{2\sigma_n^2}\right\}\right], \tag{3.3}$$

where $E[a]$ is the expectation of a, and assuming M equiprobable symbols a_k, $\forall k = 0, \ldots, M-1$. The AWGN channel adds zero-mean Gaussian noise n of variance σ_n^2, and the received noisy symbol constellation is $y = a + n$. Entropy $H(a)$ is defined as $\sum_{k=0}^{M-1} log_2(a_k)p(a_k)$. Equation 3.3 is used later in Section 3.7 to evaluate capacity for a BSPK constellation (where M=2 and $\mathbf{a} = [-1, 1]$) over the AWGN channel.

The history of error control coding, developed after Shannon's initial work, has been a search for codes which can perform up to capacity, to the very limits of what is possible in information theory. With the introduction of iteratively decoded codes such as Turbo codes [1], [2], serially concatenated convolutional codes [3] and low-density parity-check codes (LDPCs) [13], [14], [15], the goal of capacity-approaching codes has finally become a reality, some 50 years after Hamming codes [16], [17] initiated error control coding design.

3.3 Hamming codes

Richard Hamming developed the (7,4) Hamming code [16] in 1949, a simple error-correcting code that can correct a single error and detect two errors. Marcel Golay generalized the (7,4) Hamming code to an entire family of Hamming codes [17] with the same error-correcting properties, and also developed the two perfect Golay codes.

These codes are **block codes**; they encode a block of information bits of fixed length k to produce a codeword of fixed length N; such codes are denoted as (N,k) codes. The repetition code examined earlier is also a block code.

Hamming and Golay codes are also multiple **parity-check codes**. Each extra coded bit is composed of a modulo 2 sum of various combinations of the information bits, and is called a **parity** bit. The equation that describes which information bits are added to produce each parity bit is known as a **parity-check equation**. These equations are combined into a **parity-check matrix** $\mathbf{H}$, where each row $\mathbf{h}_i$ of the matrix $\mathbf{H}$ is one parity-check equation, and each column $\mathbf{h}_j$ refers to one codeword bit; a one in position h_{ij} indicates that codeword bit j is involved in the parity-check sum of parity-check equation i. Any codeword $\mathbf{x}$ satisfies all the parity-check equations of $\mathbf{H}$, that is, $\mathbf{Hx} = \mathbf{0}$ with modulo 2 addition.

The original (7,4) Hamming code had the following parity-check matrix

$$\mathbf{H} = \begin{bmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

In this code, the parity bits are bits 1, 2 and 4, while the information bits are bits 3, 5, 6 and 7. With 4 information bits, the (7,4) Hamming code has 2^4=16 codewords.

The information bits $\mathbf{u}$ are converted to an encoded sequence $\mathbf{x}$ by multiplication with a generator matrix $\mathbf{G}$ as $\mathbf{x}^T = \mathbf{u}^T\mathbf{G}$. The generator matrix $\mathbf{G}$ is found from $\mathbf{H}$ by converting $\mathbf{H}$ into systematic form, so that the last $N - k$ columns of $\mathbf{H}_{\text{sys}}$, the parity portion, form the identity matrix $\mathbf{I}$. The information portion of $\mathbf{H}_{\text{sys}}$ is termed $\mathbf{A}$, and $\mathbf{H}_{\text{sys}} = [\mathbf{A}|\mathbf{I}]$. Then $\mathbf{G}$ is found as $\mathbf{G} = [\mathbf{I}|\mathbf{A}^T]$.

Note that the parity-check matrix $\mathbf{H}$ for the original (7,4) Hamming code is not systematic. Its systematic parity-check matrix $\mathbf{H}_{\text{sys}}$ is found quickly by moving columns 1, 2 and 4 to columns 7, 6 and 5. Moving columns corresponds to exchanging information and parity bits. Thus the parity bits of the original (7,4) Hamming code are bits 1, 2 and 4, and the parity bits of the systematic form of the (7,4) Hamming code are bits 5, 6 and 7.

The original (7,4) Hamming code was decoded via **syndrome decoding**, which proceeds as follows. First, given the received noisy codeword symbols $\mathbf{y}$, a hard decision is made for the estimate of the codeword $\hat{\mathbf{x}}$. A hard decision consists of choosing a symbol $\hat{x}_i$ which is closest to y_i, that is, find x_i which maximizes the probability $p(y_i|x_i)$. When binary-phase-shift-keying (BPSK) modulation is used, meaning that the transmitted symbols are either 1 or -1, a hard decision on $\mathbf{y}$ consists of taking the sign of each received symbol, so $\hat{x}_i = \text{sign}(y_i)$.

The **syndrome** is then found by multiplying $\mathbf{H}\hat{\mathbf{x}}$. If $\hat{\mathbf{x}}$ is a valid codeword, the syndrome will equal $\mathbf{0}$. If noise is such that $\hat{\mathbf{x}}$ is not a (7,4) Hamming codeword, then one or more of the parity-check equations will sum to one, indicating an error.

A fascinating feature of syndrome decoding of the Hamming code is that if there is only one error, the value of the syndrome tells where the error is. For example, if the (7,4) Hamming codeword 0110011 is sent but the channel corrupts it to 0110111, there is an error in the fifth bit position. Multiplying $\mathbf{H}\hat{\mathbf{x}}$, using modulo 2 operations, gives 101 instead of 000, indicating an error, and the syndrome value shows that error to be in position 101 - the fifth bit position!

Hamming codes are further described by their **Hamming distance** d_H, which is the number of binary digits which differ between any two codewords. The **minimum Hamming distance** d_{min} is the minimum digit difference obtained when considering all possible two-codeword combinations of the code. Since these codes are linear, the sum of any two codewords modulo 2 is also a codeword. Thus the minimum Hamming distance may be found by considering the **Hamming weight** of each codeword, which is the sum of ones in each codeword. The minimum Hamming distance will be the minimum Hamming weight because the Hamming weight is the distance of each codeword from the all-zeros codeword (AZCW).

The minimum Hamming distance, often shortened to **minimum distance** is an important feature of an error-control code. A linear block code can detect $d_{\min} - 1$ errors and correct $\lfloor (d_{\min} - 1)/2 \rfloor$ errors. In general, the larger the $d_{\min}$ of a code, the better its performance because of its stronger error-correcting capability.

Hamming codes are typically described as $(N,k,3)$ Hamming codes as all Hamming codes can be shown to have $d_{\min}$=3. The extended Hamming codes, which have one extra parity bit composed of summing all the information bits together modulo 2, have $d_{\min}$=4, and slightly better performance. The (23,12) Golay code has $d_{\min}$=7.

3.4 Code Performance

A typical BER vs SNR plot is shown in Figure 3.1, which shows the BER performance of the (7,4,3) Hamming code and the (8,4,4) extended Hamming code compared with uncoded BPSK transmission. Both codes use maximum-likelihood (ML) decoding, discussed later, instead of hard decision syndrome decoding. BPSK transmission of the codewords is used over an AWGN channel. BPSK capacity for a rate 1/2 code over the AWGN channel is shown at 0.19 dB. This capacity limit is the best performance possible for any rate 1/2 code using BPSK modulation with an AWGN channel.

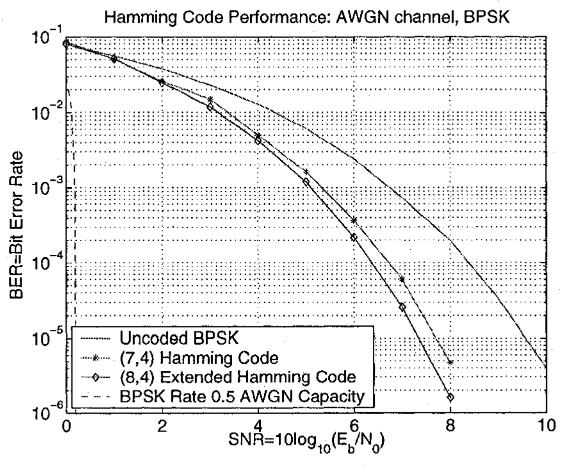


Figure 3.1: SNR vs BER Performance of (7,4) Hamming and (8,4) Extended Hamming Codes

Coding gain is the improvement in SNR gained by using a code instead of using uncoded transmission, for the same modulation and channel. From Figure 3.1, the performance of the (7,4) Hamming code at a BER of 10^{-5} occurs at SNR 7.6 dB. The (8,4) extended Hamming code performs slightly better, with a BER of 10^{-5} at SNR 7.3 dB. Uncoded BPSK modulation has a BER of 10^{-5} at SNR 9.6 dB. The (7,4) Hamming code has a coding gain of 2 dB and the (8,4) extended Hamming code a coding gain of 2.3 dB. This results in a reduction of transmitter power required to achieve the specified BER by a factor of 1.7 for the (8,4) Hamming code compared with uncoded BPSK. The (8,4) extended Hamming code performance is still 7.1 dB from BPSK capacity. The 50 years following the development of Hamming codes have seen a slow but steady push in performance towards the capacity limit. Only the last 10 years, however, have seen the final jump towards truly capacity-approaching codes with the development of large iteratively decoded codes such as Turbo codes, which are described in the following Chapter 4. Before introducing these iteratively decoded codes, a few earlier types of error-control codes relevent to this thesis are discussed.

3.5 Convolutional Codes

Convolutional codes [18] were introduced by Peter Elias in 1955. Unlike block codes which have fixed length and codewords, convolutional codes produce an ongoing stream of encoded bits given a continuous stream of information bits. The code generator may be viewed as a discrete digital filter with a finite number of unity-valued delay taps producing modulo two outputs, which are the encoded parity bits. Alternately, the code generator may be described as a finite state machine (FSM), generating encoded bits from selected sums of a finite number of delayed bits which together constitute the current state of the FSM.

The ability to encode and decode a convolutional code in real time, without waiting for an entire codeword to arrive, is a most attractive feature. Additionally, frame synchronization is not required for detection of a convolutional code, as a frame offset of a few bits results in loss of only those initial bits, rather than possibly the entire frame as can be the case for block codes. Convolutional codes have been one of the workhorses of coding applications, especially in satellite communications and telecommunications, since the late 1960s. Only in the last few years have Turbo codes and LDPCs begun to edge out convolutional codes in wireless communications and video broadcasting standards.

The strength of convolutional codes comes from the fact that each coded bit has been derived from several previous input bits. If one transmitted bit is received incorrectly, decoding can utilize the previously received bits to hopefully correct that bit.

The constraint length ν of a convolutional code refers to the number of delay elements used in the filter or FSM. Increasing the constraint length ν increases the depth of the filter and length of time in which a delayed bit influences the encoded output, but also increases the number of possible states which the encoder can be in, increasing decoder complexity. Larger constraint length, when combined with proper choice of filter taps or FSM connections, usually results in a stronger code, with better BER performance and higher $d_{\min}$ (sometimes referred to as "free distance" for convolutional codes). As is typical, this performance improvement comes at the cost of greater decoding complexity.

A convolutional encoder for a rate 1/2, ν=2, convolutional code is shown in Figure 3.2. The filter taps are expressed as polynomials in D where D is a delay element, and then represented as a binary number and converted to octal notation. Thus output $v^{(1)}$ is found as $u(1+D^2) = u(g^{(1)}(D))$, where $g^{(1)}(D)$=101=5 in octal. Output $v^{(0)}$ is $u(1+D+D^2) = u(g^{(0)}(D))$, where $g^{(0)}(D)$=111=7 in octal. Encoding is described as $v(D) = [v^{(1)}(D), v^{(0)}(D)] = u(D)g(D)$, with $g(D) = [g^{(1)}(D), g^{(0)}(D)] = [1+D^2, 1+D+D^2]$.

This encoder is represented in feedforward form, also known as **controller canonical** form because the states or filter stages are directly controlled by the inputs and easily determined. The feedforward encoder is nonsystematic. Alternately, a systematic encoder for this same code could be designed with tap polynomials of $g(D) = [1, (1+D+D^2)/(1+D)]$, so that $v^{(1)} = u$. The systematic encoder has a feedback

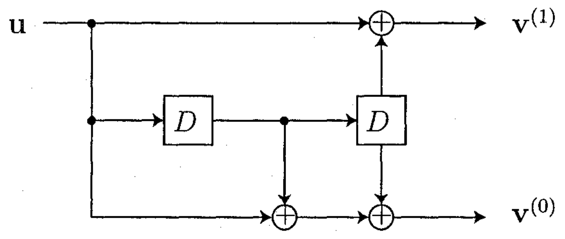


Figure 3.2: Encoder for Rate 1/2 Constraint Length 2 Convolutional Code

component.

Convolutional codes are decoded using a state trellis, which describes the topology of the code. If the code is in a certain state, a bit 0 input will move it to another state, while a bit 1 input results in a different state. The state trellis describes all possible current states and input bit values and the new state which results from them.

Figure 3.3 displays the state trellis for a rate 1/2, constraint-length ν=2 convolutional code. Because this code has constraint-length 2, or 2 delay elements in the filter or FSM, it has 4 possible states, which are shown vertically. States on the left are the current states, and they are connected with lines indicating the input bit value which moves the current state to the new state, and the output codeword bits corresponding to the current state and input bit. Since this is a rate 1/2 code, there are two codeword bits for every input bit. The FSM for this code is configured in the feed-forward form of Figure 3.2.

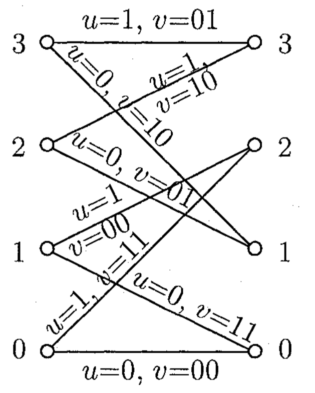


Figure 3.3: Decoding Trellis for Rate 1/2 Constraint Length 2 Convolutional Code

Decoding moves along these trellis paths. Along each path, a metric is calculated which indicates how likely that path is. A logarithmic metric commonly used for equal-energy modulation schemes like phase-shift-keying (PSK) is β_k=Re($x_k y_k^*$) where y_k is the received symbol at time k and x_k is the symbol associated with the path. Metrics along connected paths are added at each successive time interval. The path at the end with the largest metric is the most likely path, and the bits are decoded as those associated with the chosen path. This is **maximum-likelihood decoding**, because the codeword sequence with the maximum likelihood of being the correct one is chosen.

Sequential decoding [19] is an approximation to maximum likelihood decoding which used to decode convolutional codes. Several sequential algorithms have been introduced; good discussions of these algorithms are found in [20] and [21]. Long-constraint-length codes can be decoded with sequential decoding, but the method has variable decoding time, which is an implementation handicap. In 1967, **Viterbi decoding** [10] was developed by A.J. Viterbi. Viterbi decoding implemented maximum-likelihood decoding over a trellis with reduced complexity, by elimination of trellis paths which could never be part of the most likely sequence path. The Viterbi algorithm (VA) has fixed decoding time, which is convenient for implementation. Viterbi decoding of convolutional codes, either alone or serially concatenated with an outer Reed-Solomon code, has been a mainstay of deep-space and satellite communication for many years.

Convolutional codes were often serially concatenated with an outer Reed-Solomon code [22], especially in satellite communications such as the *Voyager* and *Galileo* space missions. Reed-Solomon codes are algebraic block codes designed to have a high minimum distance for their length. The purpose of the Reed-Solomon code, with its ability to correct a large number of errors, was to clean up any errors left after decoding the convolutional code. This concatenation helped to correct "burst errors" which occured when several bits in a row were received incorrectly. The convolutional code with its dependency on previous bits had more difficulty with burst errors.

In serial concatenation, the outer code serves as a wrapper code around the inner convolutional code. The information sequence is first encoded by the outer code, and the outer-encoded sequence is sent to the inner convolutional code for final encoding. Then the encoded sequence is transmitted. Decoding of concatenated codes occurs in two stages, inner first and then outer, reversing the encoding process. Any errors left after decoding the inner encoded bits are usually corrected when decoding the outer-encoded sequence.

Convolutional codes provided a big push towards capacity, before iteratively decoded concatenated codes such as Turbo codes were developed in the mid-1990s. A large-constraint-length rate 1/2 convolutional code used on the Pioneer II mission of 1973 achieved a BER of 10^{-5} at SNR 2.5 dB [21], only 2.3 dB from capacity.

3.6 Trellis Coded Modulation

In the early 1980s, trellis-coded modulation (TCM) [8], [23], [24] was developed, a concept first proposed by Wozencraft [25] and Massey [26] and later brought to its full potential by Ungerböck. TCM is an innovative coding methodology which conserves bandwidth efficiency by mapping the output of a convolutional code to a higher-order modulation scheme such as 8-PSK. A rate 2/3 convolutional code mapped to 8-PSK symbols has a transmission rate of 2 bits/symbol, the same as for QPSK. If a rate 1/2 convolutional code were used alone, and QPSK transmission used, the rate would be 1 bit/symbol, a loss in bandwidth efficiency compared to uncoded QPSK due to the rate 1/2 encoding.

However, the 8-PSK symbols are closer together than QPSK symbols, and thus have a higher error rate. At first glance, it seems that nothing has been gained, but Ungerböck's scheme does more than randomly map coded bits to symbols. TCM uses a special technique of symbol mapping called "set partitioning", which divides the symbol constellation into subsets, for example, 3 subsets for 8-PSK. Each successive subset divides the constellation further into symbols which are a greater distance apart, until the final subset is composed of sets of 2 symbols which are 180^o apart from each other. These symbols, which are a maximum distance apart, or rather the bits which map to those symbols, are then assigned to parallel branches of the convolutional code trellis. Branches coming from the same state are assigned to the next higher subset.

Set partitioning and appropriate mapping of the subsets to the trellis branches insures that the most likely trellis errors are assigned to symbols which are maximally distant from each other. In this way, the convolutional code increases the $d_{\min}$ of the TCM system. As the ML decoder chooses the most likely sequence, it is the distance between sequences which becomes important. Judicious mapping increases the minimum distance between sequences and more than compensates for the closer symbol spacing of the 8-PSK modulation. A coding gain of 5.0 dB over QPSK was obtained with a rate 2/3 constraint-length 7 convolutional code and 8-PSK mapping [23]. Even higher coding gains have been obtained using larger constraint-length convolutional codes, or higher-order modulation with higher rate codes.

TCM increases error-rate performance, or power efficiency, without sacrificing data rate, or bandwidth efficiency. Before TCM, use of error-control codes to lower error rates, or enhance power efficiency, required the transmission of extra parity symbols, which also lowered the data rate. This was a significant problem for the telephone modem industry, as voiceband modems in the late 70s were slow, limited by poor performance at high speeds. Introducing error-control coding to clean up the high-speed signal reduced the data rate and again limited modem speed.

The advent of TCM made error-control coding without a reduction in data rate available to the telephony industry. In the early 1990's, the International Telecommunications Union (ITU), then known as CCITT, introduced TCM in its V.32 and then V.34 standards [27], [28] for high-speed modems. Trellis-coded modulation was a significant contributor to the rapid ascent of high-speed voiceband modems.

3.7 Approaching Capacity

The history of error-control coding from Shannon's day to the present can be viewed as a search for codes that can provide low error rate performance as close to the Shannon capacity limit as possible. Figure 3.4 displays the Shannon limit for BPSK modulation over the AWGN channel [9], together with the SNR achieved by several error-control codes using BPSK transmission at a BER of 10^{-5}. Uncoded BPSK is also shown, with a BER of 10^{-5} at SNR 9.5 dB. Most of the BCH [29], [30] (another type of algebraic block code, similar to Reed-Solomon codes) and convolutional code results in Figure 3.4 were obtained from [21], [38].

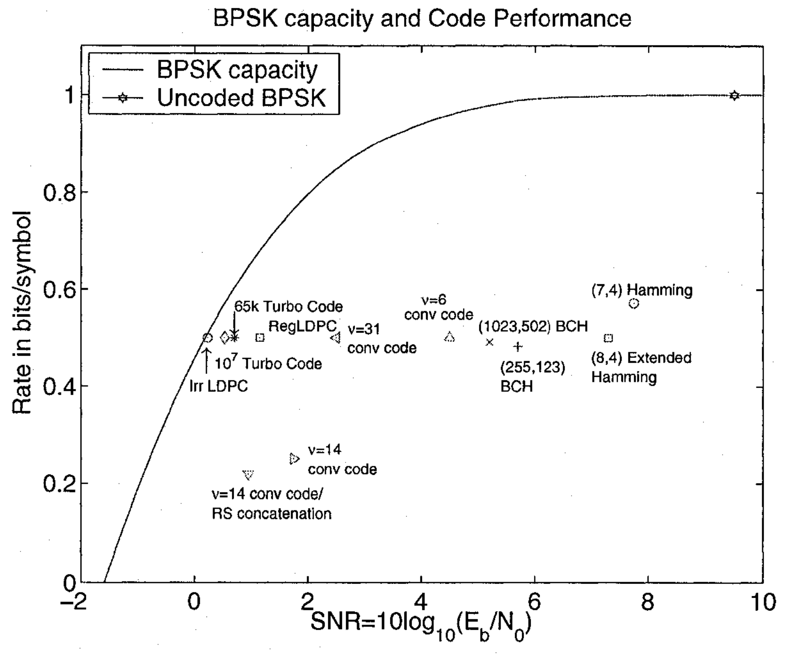


Figure 3.4: BPSK Capacity and Code Performance at BER=10^{-5}

A (255,123) BCH code decoded with algebraic decoding achieves a BER of 10^{-5} at SNR 5.7 dB [21], and improves on the (8,4) extended Hamming code by 1.6 dB. Increasing the code length to a (1023,502) BCH code results in a BER of 10^{-5} at SNR 5.2 dB [31].

Convolutional codes provided another jump towards capacity. A rate 1/2 constraint-length 6 convolutional code used in the *Voyager* mission with ML decoding reached a BER of 10^{-5} at 4.5 dB, 4.3 dB from capacity. The *Pioneer* missions used a rate 1/2 constraint-length 31 convolutional code with sequential decoding [32] to achieve a BER of 10^{-5} at 2.5 dB, a mere 2.3 dB from capacity. The *Galileo* mission used a rate 1/4 constraint-length 14 convolutional code with the biggest Viterbi decoder built to date [33] to obtain a BER of 10^{-5} at 1.75 dB [21], performance 2.5 dB from rate 1/4 BPSK capacity. Addition of an outer Reed-Solomon code concatenated with the convolutional code reduced the SNR by a further 0.8 dB [21], though slightly decreasing the code rate as well.

Trellis-coded modulation (not shown in Figure 3.4) using a constraint-length 8 convolutional code with 8-PSK modulation for a code rate of 2 bits/symbol achieved a BER of 10^{-5} at about 5.5 dB [8], a 4 dB improvement over uncoded QPSK and 2.6 dB from 8-PSK capacity at a rate of 2 bits/symbol. This distance from capacity is similar to that achieved by convolutional codes alone, but trellis-coded modulation has the advantage of a significantly higher data rate. Increasing the constraint length ν to 16 improved the SNR to 4.4 dB [21].

Turbo codes or parallel concatenated convolutional codes (PCCCs), presented in the following chapter, provided the next big jump towards capacity, with the power of iterative decoding achieving a BER of 10^{-5} at 0.7 dB for a rate 1/2 length 65,536 Turbo code [1]. Increasing the Turbo code length to 10^6 bits lowered the SNR to 0.54 dB ([34], Figure 2).

Low-density parity-check codes (LDPCs) [13], [14], originally developed by Gallager in 1962 and later rediscovered by MacKay and Neal, then extended to irregular form by Luby *et al* [35], [36] and Richardson and Urbanke [34], supplied the final push towards truly capacity-achieving codes, though not in their original, regular design. Regular (3,6) rate 1/2 LDPCs with a block length of 10^6 reached a BER of 10^{-5} at 1.15 dB ([34], Figure 2), about 0.6 dB away from rate 0.5 Turbo codes. However, an irregular rate 0.5 LDPC with some variable degrees up to 200 and length 10^7 achieves a BER of 10^{-5} at SNR 0.23 dB ([34], Figure 2), a mere 0.04 dB from capacity! Increasing the variable degrees up to 8000 decreases the threshold, or SNR onset of low BER for an infinitely long code of this degree structure, to 0.0045 dB from capacity [37]; however, this code is too large and has too high variable node degrees to be feasible for implementation.

Chapter 4

Turbo Coding and Iteratively-Decoded Codes

Turbo codes [1], [2] were invented in 1993 by Berrou, Glavieux and Thitimajshima. They represented a major step forward in capacity-approaching codes, coming within fractions of a dB of the Shannon limit for BPSK capacity.

Turbo codes have a very characteristic bit error rate (BER) curve, as shown in Figure 4.1, which shows the BER performance of the original rate 1/2 length 65,536 turbo code introduced in [1]. This turbo code is composed of two component rate 1/2 memory 4 recursive systematic convolutional codes.

The BER curve of an iteratively-decoded code can be divided into three portions. There is a high BER region, where the turbo code fails at very low SNR, resulting in high BER. The turbo cliff or waterfall region follows, where the BER performance drops precipitously within fractions of a dB increase in SNR to very low error rates. Finally, at high SNR, there is often an error floor or error flare where the BER curve flattens out. This error floor is due to the presence of low-minimum-distance error events, described in greater detail in Chapter 6.1, that dominate at high SNR.

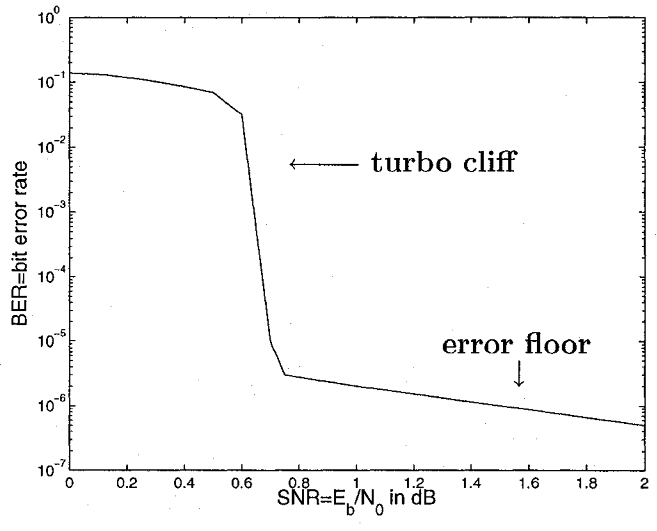


Figure 4.1: Original Turbo Code BER Performance [1], [21]: rate 1/2 length 65,536 turbo code, BPSK modulation, AWGN channel, 18 iterations.

4.1 Parallel Concatenated Codes

The original turbo code was a parallel concatenated convolutional code (PCCC) consisting of two identical recursive systematic convolutional codes. These two codes are separated by an interleaver Π, that permutes the information sequence $\mathbf{u}$ to $\mathbf{u}'$ according to $\mathbf{u}' = \Pi(\mathbf{u})$. Both codes see the same information bits $\mathbf{u}$. However, the information sequence of one is permuted by the interleaver to $\mathbf{u}'$, so the parity bits $\mathbf{p}_1$ and $\mathbf{p}_2$ are different. The information sequence $\mathbf{u}$ and the two coded parity sequences $\mathbf{p}_1$, $\mathbf{p}_2$ are transmitted as $[u_1, p_{1,1}, p_{2,1}, u_2, p_{1,2}, p_{2,2}, \ldots, u_N, p_{1,N}, p_{2,N}]$ where $p_{i,j}$ indicates the jth symbol of parity sequence p_i from encoder i. This sequence has rate 1/3.

To increase the rate, the parity sequences may be punctured, so that alternating symbols from each sequence are sent, for example, $[u_1, p_{1,1}, u_2, p_{2,2}, u_3, p_{1,3}, u_4, p_{2,4}, \ldots]$ giving rate 1/2. A block diagram of the PCCC encoder is shown in Figure 4.2.

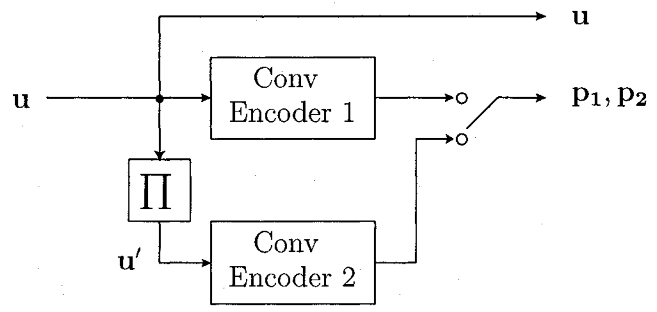


Figure 4.2: Encoder for parallel-concatenated convolutional code (PCCC) with puncturing.

4.2 Turbo Decoding

The turbo decoding system is the key to the fantastic performance of turbo codes. Maximum likelihood (ML) decoding over all possible code sequences is not practical, as the number of possible code sequences is 2^k. The information length k is typically large, at least 100 and often 10^3 or 10^4 bits. The original turbo code had interleaver length and information length $k = 65,536$ bits.

Instead, a sub-optimal method is considered, where soft-decision maximum *a posteriori* (MAP) decoding is performed for each constituent code. The decoders are separated by an interleaver/deinterleaver as appropriate. MAP decoding is also known as APP decoding, because it outputs *a posteriori* probabilities as soft information. Throughout this thesis, the term 'APP decoder' will be used, which is interchangeable with 'MAP decoder'. The term 'soft information' refers to probabilistic information, rather than discrete hard decisions.

If only a single pass through each decoder was used, the performance of the PCC would be significantly inferior to ML decoding. However, because soft-decision APP decoders are used, which utilize *a priori* information as well, decoding can become iterative. Each APP decoder passes to the other its output soft *a posteriori* probability (APP) information, to be used as improved input *a priori* information. Each iteration improves decoding performance until convergence to an estimated information sequence occurs, often within 10 iterations. The purpose of the interleaver Π is to eliminate correlation between neighboring symbols into the next decoder. Figure 4.3 displays the PCCC turbo decoding process, explained in greater detail below.

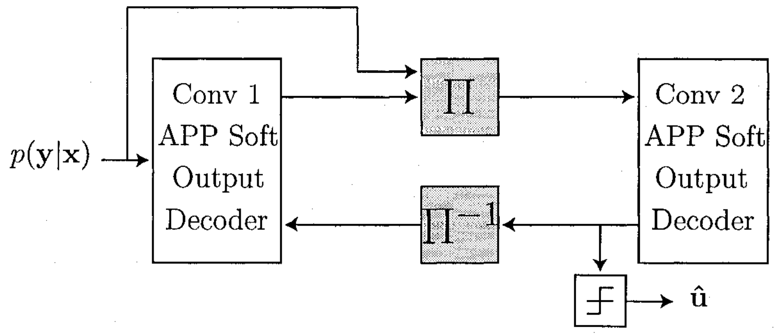

$p(\mathbf{y}|\mathbf{x})$
Conv 1 APP Soft Output Decoder
Π
Π^{-1}
Conv 2 APP Soft Output Decoder
$\hat{\mathbf{u}}$

Figure 4.3: Iterative decoder for parallel concatenated convolutional code (PCCC).

Assuming an additive white Gaussian noise (AWGN) channel, and coherent decoding with perfect phase and timing synchronization, the received noisy channel symbols are found as $\mathbf{y} = \mathbf{x} + \mathbf{n}$, where $\mathbf{n}$ is a zero-mean, circularly-symmetric, Gaussian-distributed random variable. The received sampled channel values y_k come from the receiver's matched filter and sampler. The channel values $\mathbf{y}$ are converted to channel metrics

$$p(y_k|x_k = X_m) = \frac{\exp\left(-\frac{|y_k - X_m|^2}{N_0}\right)}{\pi N_0}, \quad \forall m = 0, \dots, M-1, \tag{4.1}$$

where M is the cardinality or alphabet size of x, and X_m are the M possible symbol values of x, such that $X_m = \exp(j2\pi m/8)$. If BPSK modulation is used, so the transmitted symbols $x \in [-1, 1]$, then the metric in probability form $[p(y_k|x_k = -1),\ p(y_k|x_k = 1)]$ may be converted to log likelihood ratio (LLR) form, as

$$\lambda_k = \log\left(\frac{p(y_k|x_k = 1)}{p(y_k|x_k = -1)}\right) = \frac{2y_k}{N_0}. \tag{4.2}$$

These channel LLRs or probabilities are sent to a soft-decision APP decoder, along with *a priori* LLRs or probabilities from the other APP decoder, if available. The APP decoder operates on the code's trellis using an algorithm such as the BCJR [39], also known as the forward-backward, algorithm. The APP decoder outputs soft APP values $p(x_k|\mathbf{y})$ (or their corresponding LLR values) A hard decision on $\hat{x}_k$ may be made at any time according to the MAP decision rule

$$\hat{x}_k = \arg\max_{x_k} p(x_k|\mathbf{y}). \tag{4.3}$$

This reduces to $x_k = \text{sign}(\lambda_k)$ for LLR values. MAP decoding reduces the bit error rate (BER) by maximizing the bit (or symbol) probability. Maximum likelihood (ML) decoding maximizes the sequence probability, which may or may not coincide with the same symbols chosen according to MAP principles.

The APP decoder is termed soft-output or soft-decision because the output probabilities are real values from [0,1], or LLR values from $[-\infty, +\infty]$ or more practically from $[-\lambda_{\max}, \lambda_{\max}]$, where $|\lambda_{\max}|$ is a maximum LLR threshold limit. A hard-decision decoder, such as a Viterbi decoder, outputs actual hard bit/symbol values as decisions. The probability or LLR values in and out of the APP decoder are called soft information; thus the APP decoders for iterative decoding are often termed soft-in soft-out (SISO) decoders.

At the beginning of the first iteration, decoder 1 receives the channel metrics for the received noisy information sequence $\mathbf{y}_0$ and noisy parity bits $\mathbf{y}_1$ from code 1 as input. No *a priori* information is available from decoder 2, so *a priori* input to decoder 1 is set to uniform probabilities (or LLR=0). Decoder 1 outputs APP values for $\mathbf{u}, \mathbf{p}_1$. Extrinsic LLRs or probabilities are sent to decoder 2 through the interleaver. The term **extrinsic** refers to the fact that the *a priori* information from the other decoder, which was initially used in calculating the APP soft information, is removed from the APP values before sending that information back to the other decoder. The purpose of removing the *a priori* information is to prevent its over-accumulation at the decoder; the other decoder originally sent the *a priori* information, and does not need to see it again.

Extrinsic LLRs are calculated by subtracting off the *a priori* LLRs from the APP LLRs; extrinsic probabilities are calculated by dividing out the *a priori* probabilities from the APP probabilities, as

$$\lambda_e(\mathbf{u}) = \lambda_{\text{APP}}(\mathbf{u}) - \lambda_a(\mathbf{u}); \tag{4.4}$$

$$p_e(u_k) = p_{\text{APP}}(u_k|\mathbf{y})/p_a(u_k), \tag{4.5}$$

where λ_e and p_e are the extrinsic LLRs and probabilities, λ_{APP} and p_{APP} are the output APP LLRs and probabilities, and λ_a and p_a are the input *a priori* LLRs and probabilities.

This subtraction to obtain extrinsic LLRs is shown in the serial concatenated decoder of Figure 4.5. The same extrinsic operation is performed in turbo decoding, but to focus instead on the iterative loop between the two component APP decoders, is not shown in the turbo decoder of Figure 4.3.

The extrinsic values output from decoder 1 for $\mathbf{u}$ are sent through the interleaver Π to decoder 2 as *a priori* $p_a(\mathbf{u}')$. Decoder 2 receives these *a priori* values in addition to interleaved channel metrics corresponding to $\mathbf{y}_0'$ and $\mathbf{y}_2$. Decoder 2 calculates APP values for $\mathbf{u}', \mathbf{p}_2$, and sends APP values for $\mathbf{u}'$ through the deinterleaver to decoder 1 to be used as a priori $p_a(\mathbf{u})$. This constitutes one iteration. Decoding continues in this fashion for either a fixed number of iterations or until a convergence criteria is met.

4.3 Serially-Concatenated Codes

Iterative decoding principles for turbo decoding were extended to serially-concatenated codes (SCCs) by Benedetto, Divsalar, Montorsi, and Pollara in 1998 [3]. Again, two codes are concatenated together, this time serially, with an interleaver Π separating them to reduce bit correlation. Figure 4.4 shows a general serially-concatenated encoding system. The first code is termed the **outer code**, and maps the information sequence $\mathbf{u}$ into the encoded sequence $\mathbf{v}$. An interleaver $\Pi(\mathbf{v}) = \mathbf{v}'$ permutes $\mathbf{v}$ into the scrambled sequence $\mathbf{v}'$ to break up correlation between the bits. The second code maps $\mathbf{v}'$ into the coded sequence $\mathbf{x}$, which is then transmitted across the channel to the receiver. This second code is termed the **inner code**.

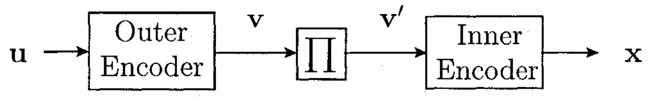


Figure 4.4: Encoder for serially-concatenated code (SCC).

Decoding follows iterative turbo decoding principles. The main difference is that only the inner code of the SCC sees the channel and thus only the inner APP decoder receives channel metrics. The outer APP decoder receives only extrinsic information from the inner decoder as *a priori* input. Also the inner APP decoder sends its information APP values (deinterleaved) to the outer decoder, while the outer APP decoder sends its coded interleaved APP values to the inner decoder. APP decoders generate APP values for both information and coded sequences; the PCCCs only utilize the information sequence values. A block diagram of the serially concatenated decoding system is shown in Figure 4.5.

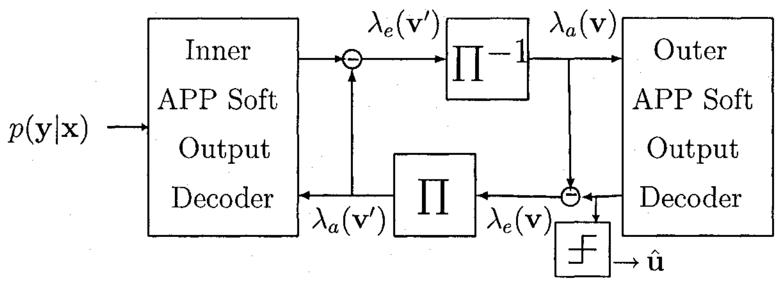


Figure 4.5: Iterative decoder for serially-concatenated code (SCC).

Chapter 5

System Description

5.1 Introduction

The goals of the system presented in this thesis are: near-capacity performance at a high system rate, with good compromise between turbo cliff onset and error floor, and the ability to incorporate channel phase and timing estimation within the iterative decoder. Additionally, the system design should emphasize ease of implementation and minimal design area.

To achieve near-capacity performance, an iteratively-decoded concatenated coding system is required. High throughput or system rate indicates use of higher order modulation. Use of a differential inner modulation code would provide the potential for operation in the presence of channel phase offset. Simple component codes result in easily designed encoders and decoders, and as will be shown in Section 6.2, also result in capacity-approaching concatenated systems.

Therefore, this thesis considers a serially-concatenated encoding system composed of an outer rate 2/3 binary error-correction code and an outer differential 8-PSK modulation code, separated by an interleaver. Differential modulation provides rotational invariance, which is a significant asset when decoding in the presence of a channel phase offset. Rotational invariance is considered further in Section 5.3. Another coding option that provides rotational invariance to specific phase angles is a rotationally-invariant trellis code such as in [40]. However, these rotationally-invariant codes are only invariant to specific phase angle rotations, not any possible phase offset angle. When differentially decoded, differential modulation is invariant to all possible phase angle rotations, though at the cost of performance loss with respect to coherent decoding, where the phase offset is corrected before decoding is performed. Without differential decoding, the differential 8-PSK trellis still provides some initial symbol information even in the presence of a phase offset halfway between constellation angles. Section 9.3 shows this ability of the differential 8-PSK decoder when operating with a range of uncorrected constant channel phase offsets. The differential 8-PSK trellis is of course rotationally invariant to all multiples of $\pi/4$ radians.

Differential modulation can be viewed as a recursive rate 1 convolutional code [41], and thus it functions as a viable inner code in an SCC, providing interleaver gain [3]. Differential modulation has been previously used in combination with an outer rate 1/2 convolutional code as binary SCCs with a 4-state convolutional code and DQPSK in [41] and a 16-state convolutional code and DBPSK in [42], providing good results when coherently decoded.

The differential modulation code may also be viewed as a mod-8 phase accumulator. Differentially-encoded turbo-coded modulation shares structural similarities with repeat-accumulate (RA) [43] codes. RA codes are serially-concatenated codes consisting of a repetition code separated by a bit interleaver from an accumulate code (for BPSK modulation, this consists of adding successive bits mod 2). These simple codes have also shown excellent near-capacity performance when iteratively decoded.

The iterative decoding system used in this thesis is comprised of the component decoders, with the inner modulation code decoded using MAP decoding, as a recursive convolutional code, instead of differential demodulation. Iterative decoding will proceed in turbo fashion.

5.2 Encoding System

The proposed serially concatenated differentially-encoded turbo-coded modulation system is shown in Figure 5.1. The outer code is a rate 2/3 binary error-control code, such as a (3,2,2) parity code or a rate 2/3 convolutional code. A sequence of information bits $\mathbf{u} = [u_0, \ldots, u_{K-1}]$ of length k is encoded by the outer rate 2/3 code into a sequence of coded bits $\mathbf{v} = [v_0, \ldots, v_{3k/2-1}]$ of length $3k/2$. A moderate length of k is considered for most simulations in this thesis, with $k = 10,000$. For comparison, the original turbo code, a parallel-concatenated convolutional code, had an interleaver size of 65,536 bits, and thus an information sequence length k of 65,536 bits.

If a (3,2,2) parity check code is used for the outer code, implementation of the outer encoder is very simple. The (3,2,2) parity check code is a block code that takes two data or information bits, and generates a parity bit based on the modulo 2 sum or exclusive-OR (XOR) of the two data bits. Thus this code has 4 possible codewords: 000, 011, 101, 110. The (3,2,2) parity code has a Hamming distance $d_H = 2$ between any two different codewords, and $d_{\min} = 2$.

The coded bits $\mathbf{v}$ are interleaved or permuted bitwise through a random interleaver [2],[3]. An interleaver Π permutes the position of a bit v_i to position $\Pi(v_i)$ in a one-to-one mapping. Chapter 8 discusses interleaver designs specifically designed to improve performance, but for now, randomly-chosen interleaver permutations are assumed, as representative of the average code performance.

These interleaved coded bits $\mathbf{v}'$ are mapped to 8-PSK symbols $\mathbf{w} = [w_0, \ldots, w_{k/2-1}]$, $w_k = \sqrt{E_s}e^{j\theta_k}$. The symbol energy is normalized to $E_s = 1$, and the 8-PSK symbols have unit magnitude and phase angle $\theta_k \in [0, \pi/4, \ldots, 7\pi/4]$ radians. Initially, natural mapping is used, which labels the bit combinations $000 - 111$ increasing consecutively counterclockwise from 0 radians.

The 8-PSK symbols are then differentially encoded, so that the current transmitted differential symbol $x_i = w_i x_{i-1}$. This differential 8-PSK encoding serves as the inner code of the serially concatenated system, and is viewed as a recursive non-systematic convolutional code [41]. A recursive inner code is necessary to provide interleaving gain in a serially concatenated system [3]. Transmission of a block of encoded symbols is initialized with $x_0 = w_0$, and terminated with a reference symbol $x_{k/2} = \sqrt{E_s}e^{j0}$ to end the coded sequence in the zero state.

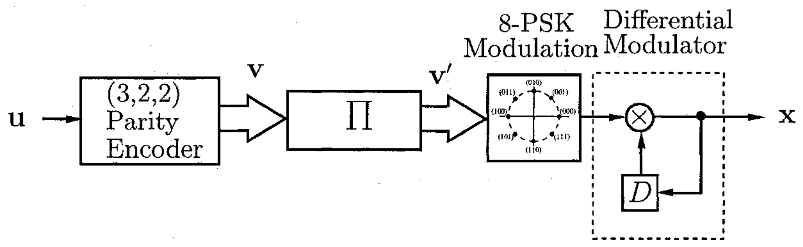


Figure 5.1: Serial turbo encoder for differential turbo-coded modulation.

5.3 Differential 8-PSK Trellis

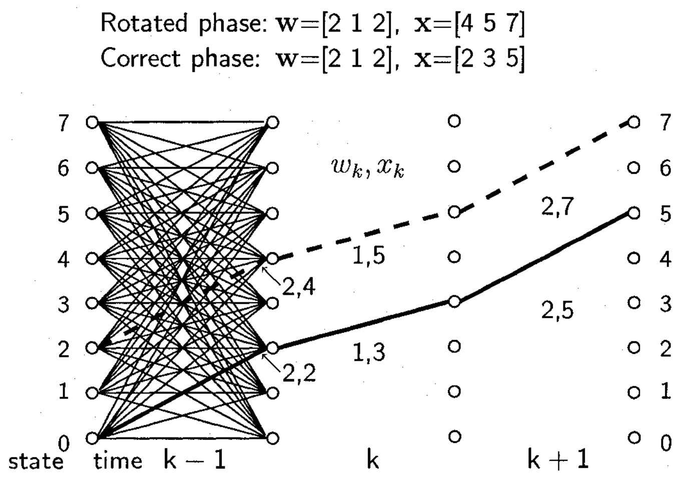
The differential 8-PSK inner code, when viewed as a recursive, non-systematic rate 1 convolutional code, has a regular, fully-connected 8-state trellis. The trellis states correspond to the current transmitted differential symbol. Thus any trellis decoding algorithm, for example the BCJR algorithm, may be used to decode this inner code, instead of using differential demodulation.

A simplified three-stage differential 8-PSK trellis section (corresponding to three symbol time intervals) is shown below in Figure 5.2. The actual trellis is fully connected but for clarity, only the first stage shows all connections. Two decoding paths are depicted in the trellis section. The solid bold path indicates the correct path, with the dashed bold path above it resulting from a phase rotation (due either to a channel phase offset or incorrect phase synchronization between transmitter and receiver) of $\pi/2$ radians. Both the original 8-PSK symbols $\mathbf{w}$ and the transmitted D8-PSK symbols $\mathbf{x}$ are shown as w, x above each corresponding trellis branch. The correct (solid bold) 8-PSK information symbol sequence is $\mathbf{w} = e^{j90^o}, e^{j45^o}, e^{j90^o}$ and the correct differential symbol sequence is $\mathbf{x} = e^{j90^o}, e^{j135^o}, e^{j225^o}$. The trellis states correspond to the new differential symbol on the incoming branch.

If the channel rotates the transmitted sequence by a multiple of $\pi/4$ rads, in this case $\pi/2$, the decoded sequence shifts to another path parallel to the correct path, assuming no noise. The decoded sequence (dashed bold) corresponds to the information symbol sequence $\hat{\mathbf{w}} = e^{j90^o}, e^{j45^o}, e^{j90^o}$ and the differential symbol sequence $\hat{\mathbf{x}} = e^{j180^o}, e^{j225^o}, e^{j315^o}$.

Note that while the $\pi/2$ phase rotation affects the transmitted D8-PSK symbols $\mathbf{x}$, the information 8-PSK symbols $\mathbf{w}$ are identical on both paths. Thus the parallel, rotated path still provides the correct information sequence! The differential trellis and code are **rotationally invariant** to multiples of $\pi/4$ rads $= 45^o$ phase offset. Parallel paths in the trellis are equivalent, with respect to the information sequence $\mathbf{w}$. This rotational invariance will be significant when later considering decoding without phase synchronization.

The rotational invariance of the differential inner code means that if an entire codeword sequence is offset by a multiple of $\pi/4$, assuming no noise, the D8-PSK decoder will send the correct probabilities for $\mathbf{w}$ to the outer decoder.



Rotated phase: w=[2 1 2], x=[4 5 7]
Correct phase: w=[2 1 2], x=[2 3 5]
7
6
5
4
3
2
1
0
state
time k − 1
k
k + 1
w_k, x_k
2,4
2,2
1,5
1,3
2,7
2,5

Figure 5.2: Differential 8-PSK trellis section, 3 time stages.

5.4 Channel Model

A complex AWGN channel model with noise variance $N_0/2$ in each real dimension is used. Initially, the channel phase is assumed known at the receiver; timing and carrier synchronization are also assumed. This assumption of perfect synchronization is known as coherent decoding. With coherent decoding, the received sampled signals $\mathbf{y}$ consist of the transmitted D8-PSK-encoded symbols plus complex noise of variance N_0, *i.e.*, $\mathbf{y} = \mathbf{x} + \mathbf{n}$.

The differentially-encoded turbo-coded modulation system described here, with a rate 2/3 binary outer code and a rate 1 D-8PSK modulated inner code which maps 3 bits to 1 symbol, has a rate of 2 bits per symbol.

In Chapter 9, the coherent decoding assumption is dropped, and an unknown phase offset at the receiver is considered. The channel now consists of complex AWGN $\mathbf{n}$ plus a possibly time-varying symbol phase offset $\varphi(t)$: $y(t) = x(t)e^{j\varphi(t)} + n(t)$. Assuming the matched filter output is sampled every T seconds, where T is the symbol timing, the decoder receives as input $y(kT) = x(kT)e^{j\varphi(kT)} + n(kT)$, which we denote as $y_k = x_k e^{j\varphi_k} + n_k$.

5.5 Iterative Decoding

Decoding of this serially concatenated system proceeds iteratively according to turbo decoding principles [2], [3] as described in Chapter 4. Figure 5.3 displays a block diagram of the decoding process. Two APP decoders are used, one for each component code, as for a generic iteratively-decoded serially-concatenated decoder. Note that differential demodulation is never used at the receiver for detection in this system.

Assuming coherent decoding, received noisy two-dimensional channel symbols $\mathbf{y}$ are converted to channel metrics as

$$p(y_k|x_k) = (\pi N_0)^{-1} e^{-|y_k - x_k|^2/N_0}. \tag{5.1}$$

These channel metrics are input to the inner differential APP decoder, along with *a priori* information $p_a(\mathbf{w})$ from the outer decoder. No *a priori* information is available for the inner APP decoder during the first iteration, thus uniform *a priori* probabilities are used; $p_a\left(w_k = e^{j2m\pi/8}\right) = 1/8, \forall m = 0, \ldots, 7$. The inner APP decoder uses the BCJR [39] or

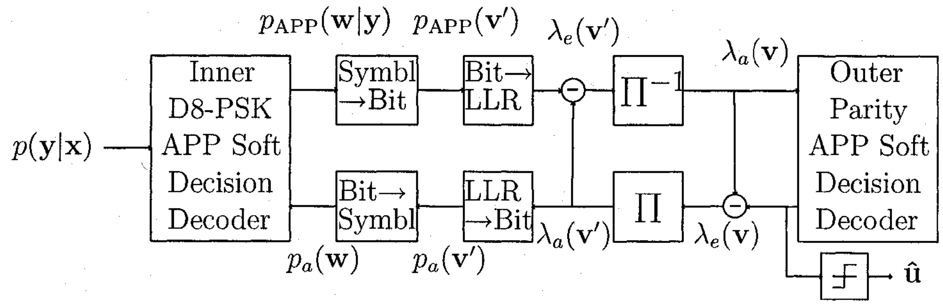


Figure 5.3: Differentially-encoded turbo-coded modulation decoder, D8-PSK inner decoder and (3,2,2) outer parity decoder.

forward-backward algorithm operating on the 8-state trellis of the differential 8-PSK code, depicted in Figure 5.2, to calculate conditional symbol probabilities on both the 8-PSK symbols $\mathbf{w}$ and the transmitted D8-PSK symbols $\mathbf{x}$.

The APP 8-PSK symbol probabilities $p_{\text{APP}}(\mathbf{w})$ are converted first to APP bit probabilities $p_{\text{APP}}(\mathbf{v}')$ through marginalization, for example,

$$p_e(v_i' = 0) = \sum_{w_k : v_i' = 0} p_e(w_k(v_i' = 0, v_{i+1}', v_{i+2}')). \tag{5.2}$$

The APP bit probabilities are then converted to APP log-likelihood ratios (LLRs) $\lambda_{\text{APP}}(v_i') = p_{\text{APP}}(v_i' = 1)/p_{\text{APP}}(v_i' = 0)$. Extrinsic output LLRs $\lambda_e(\mathbf{v}')$ are obtained by subtracting the *a priori* LLRs $\lambda_a(\mathbf{v}')$ from the APP LLRs $\lambda_{\text{APP}}(\mathbf{v}')$, as

$$\lambda_e(\mathbf{v}') = \lambda_{\text{APP}}(\mathbf{v}') - \lambda_a(\mathbf{v}'). \tag{5.3}$$

Deinterleaving the extrinsic LLRs $\lambda_e(\mathbf{v}')$ provides *a priori* LLRs $\lambda_a(\mathbf{v})$ as input to the outer APP decoder. The outer APP decoder for a simple binary code may be implemented with low complexity; the (3,2,2) parity code examined herein is simple enough, with only three coded bits and four codewords, to implement the six APP equations explicitly. The APP probabilities that express the parity constraints of the code are given by

$$\begin{aligned} p_{\text{APP}}(v_1 = 0) &\propto p_a(v_1 = 0) \cdot (p_a(v_2 = 0)p_a(v_3 = 0) + \\ &\qquad p_a(v_2 = 1)p_a(v_3 = 1)), \qquad (5.4) \\ p_{\text{APP}}(v_1 = 1) &\propto p_a(v_1 = 1) \cdot (p_a(v_2 = 0)p_a(v_3 = 1) + \\ &\qquad p_a(v_2 = 1)p_a(v_3 = 0)), \qquad (5.5) \end{aligned}$$

and analogously for v_2 and v_3. APP LLRs $\lambda_{\text{APP}}(\mathbf{v})$ are computed from these, and extrinsic LLRs $\lambda_e(\mathbf{v})$ obtained by subtracting off the *a priori* LLRs $\lambda_a(\mathbf{v})$ from the APP LLRs.

If a rate 2/3 convolutional code is used as the outer code, then APP decoding is implemented using the BCJR algorithm on the code trellis. Extrinsic LLRs are obtained as above.

These extrinsic LLRs $\lambda_e(\mathbf{v})$ are then interleaved to obtain new *a priori* LLRs $\lambda_a(\mathbf{v}')$. These must be converted first to bit probabilities $p_a(\mathbf{v}')$, then to *a priori* symbol probabilities $p_a(\mathbf{w})$ for use in the next iteration of decoding in the inner APP symbol decoder. Symbol probabilities are calculated as the product of their component bit probabilities, that is,

$$p_a(w(v_1 = V1, v_2 = V2, v_3 = V3)) = \prod_{i=1}^{3} p_a(v_i = Vi), Vi \in [0, 1]. \qquad (5.6)$$

In this fashion, with inner and outer APP decoders exchanging extrinsic information each iteration, iterative decoding continues until convergence or a fixed number of iterations is reached, at which time a hard decision on the estimated information sequence $\hat{\mathbf{u}}$ is made, based on the sign of the APP information bit LLRs $\lambda_{\text{APP}}(\mathbf{u})$ from the outer binary APP decoder.

Chapter 6

Code Properties and Performance Analysis

As mentioned in Chapter 4, iteratively-decoded concatenated code performance may be divided into three regions. In the first, the low SNR region, the BER is quite high and iterative decoding has minimal effect. The turbo cliff or waterfall region follows, where the BER performance drops sharply to low values within only fractions of a dB in SNR. Finally, at high SNR, there may be an error floor or flare where the BER curve flattens out due to the predominance of low-weight error events.

The latter two regions, the turbo cliff and error floor regions, require separate methodologies to analyze concatenated code performance. Extrinsic information transfer, or EXIT, analysis [4], [5] is used to accurately predict turbo cliff onset SNR. In this method, mutual information serves as a reliability measure of the soft information into and out of each component decoder.

The second method is the minimum distance asymptote approximation [44] of the error performance in the high SNR error floor region, where performance of turbo coded systems flattens out by following the error curve of the minimum-weight sequence error events, which become the most likely error events at high SNR.

Both methods are used to demonstrate the behavior of serially-concatenated coded modulation and to improve system performance. Firstly, the minimum distance of the differentially-encoded turbo-coded modulation system is examined, with the intention of improving error floor performance via the 8-PSK bit mapping.

6.1 Distance Spectrum Analysis

Distance spectrum analysis is a technique that provides information about the high SNR, low BER region and can asymptotically predict values for the expected error floor. This technique was developed in the days of convolutional coding and trellis coding, and has been extended to turbo codes [45] and to serially concatenated codes [3].

To understand this concept, let us first examine the concept of minimum squared Euclidean distance (MSED) and the code trellis. A convolutional code trellis section is **regular**, so that each section of the trellis corresponding to one symbol/time interval looks identical to any other section. The exceptions are the beginning and end sections of the trellis; the decoder will be initialized to the zero state, and thus the beginning few sections will have fewer branches and states due to starting in the zero state. Similarly, if tail bits are added to the convolutional code to ensure termination in the zero state (which improves code performance), the final tail sections will also have fewer branches and states due to tailing or flushing to end in the zero state. Each branch of the trellis from one state to another has a transmitted coded symbol and an uncoded symbol associated with it. A codeword corresponds to a particular sequence of branches through the trellis.

If an error is made in decoding the trellis, so that a different codeword with a different path through the trellis is chosen as the most likely codeword instead of the actual codeword, the distance between those two trellis paths is termed the Euclidean distance d. The squared Euclidean distance (SED) d^2 between the two codeword paths may be calculated as the sum of the squared distances between each branch symbol. A simple example for differential 8-PSK modulation is when the symbols [0,0,0] are sent but [0,2,6] are decoded. A 6 indicates the 8-PSK symbol $e^{j3\pi/2}$, and in general, m indicates the 8-

PSK symbol $e^{jm\pi/4}$. The squared Euclidean distances between each symbol are [0,2,2]. The SED between the actual codeword and the erroneous codeword is $d^2 = 2 + 2 = 4$. The probability of choosing the erroneous path $\mathbf{e} = [0, 2, 6]$ when the correct path was $\mathbf{c} = [0, 0, 0]$ is found according to the pairwise error probability (PEP) as

$$P_{c \to e} = Q\left(\sqrt{d^2 \frac{E_s}{2N_0}}\right), \tag{6.1}$$

where $Q(x)$ is the Gaussian Q-function, defined as

$$Q(x) = \frac{1}{\sqrt{2\pi}} \int_x^\infty \exp\left(-\frac{\beta^2}{2}\right) d\beta. \tag{6.2}$$

The Q-function may be approximated by overbounding, resulting in an approximation for the PEP as

$$Q\left(\sqrt{\frac{d^2}{2N_0}}\right) \le \frac{1}{2} \exp\left(-\frac{d^2}{4N_0}\right) \tag{6.3}$$

assuming $E_s = 1$ for equal-magnitude symbols.

The probability of choosing an incorrect path e instead of c decreases exponentially as its SED d^2 from c increases. The most probable errors will be those due to the closest erroneous paths with minimum SED from the incorrect path. The minimum SED (MSED) is denoted by $d^2_{\min}$.

Error analysis for a convolutional or trellis-based code examines all possible erroneous paths. If the code is linear, and therefore the sum of codewords is also a codeword, we may compare the distance between each codeword and the zero codeword instead of comparing all codewords with all other codewords. The average error event probability, normalized with time, is overbounded using the union bound as the sum of all error events to obtain

$$P_e \le \sum_{d_i^2} A_{d_i^2} Q\left(\sqrt{d_i^2 \frac{E_s}{2N_0}}\right) \tag{6.4}$$

where d_i^2 is the SED between codewords and $A_{d_i^2}$ is the average multiplicity of d_i^2, the average number of times d_i^2 occurs. The set of values $(d_i^2, A_{d_i^2})$ is known as the distance spectrum of the code. Details concerning this bound and its derivation may be found in [21] and similar. For purposes of examining high SNR performance of the concatenated system and error floor analysis, we are only concerned with the low weight error paths, i.e. codewords with MSED. At high SNR, the low amounts of noise make the minimum distance error paths by far the most likely error paths, so MSED error events dominate at high SNR. They cause the error floor or error flare often seen in turbo coded systems, as the higher weight error events become less and less probable and the error probability asymptotically approaches

$$P_e \sim A_{d^2_{\min}} Q\left(\sqrt{d^2_{\min}\frac{E_s}{2N_0}}\right) \tag{6.5}$$

The extension of the distance spectrum technique to turbo codes is somewhat complex, involving random interleaver analysis, and will not be covered in this thesis. A similar result to Equation 6.4 is derived, that incorporates both inner and outer codes. At high SNR, the error probability is dominated by the combined MSED error events of the inner decoder and $d_{\min}$ error events of the outer decoder, as the error probability is inversely proportional to the MSED and $d_{\min}$ due to the Q-function of equations 6.2 and 6.3. Rather than directly trying to determine the error floor value, this chapter is concerned with ways to lower the decoding error floor. The error floor is due to MSED error events, and increasing the MSED of a code lowers that error floor.

6.1.1 Minimum Distance Analysis

The trellis of a differential 8-PSK encoder is fully connected, so that the shortest error event is always only two branches long. This is because, no matter which incorrect D8-PSK symbol is associated with the first diverging branch, another D8-PSK symbol always exists which will bring the error event back to the correct state in the following merging branch.

With respect to the all-zeros sequence, the seven possible two-branch error events are

Error Event #	Diverging Branch 8-PSK (Bits) - D8-PSK	Merging Branch 8-PSK (Bits) - D8-PSK	SED	# Bit Errors
1	1 (001) - 1	7 (111) - 0	0.586	4
2	2 (010) - 2	6 (110) - 0	2	3
3	3 (011) - 3	5 (101) - 0	3.14	4
4	4 (100) - 4	4 (100) - 0	4	2
5	5 (101) - 5	3 (011) - 0	3.14	4
6	6 (110) - 6	2 (010) - 0	2	3
7	7 (111) - 7	1 (001) - 0	0.586	4

Table 6.1: Two-branch error events with associated bits, 8-PSK symbols, D8-PSK symbols, SED and number of bit errors, for naturally-mapped D8-PSK, with reference to correct all-zeros sequence.

listed for naturally-mapped D8-PSK in Table 6.1. The 8-PSK and differential 8-PSK symbols associated with each branch of the two-branch error event are shown. The SED, assuming unit energy symbols, is also given for each two-branch error event, as well as the corresponding number of bit errors.

Noting that merging branches all carry the same output symbol, only the diverging branch contributes to the minimum squared Euclidean distance (MSED). The two-branch error event MSED is simply that of naturally mapped 8-PSK, 0.586, found in error events (1) and (7). Without an outer code, the input sequences to the D8-PSK encoder are unconstrained, and the D8-PSK MSED is 0.586.

The 8-PSK mapping is not regular, *i.e.*, all symbols separated by a given squared Euclidean distance (SED) do not have the same Hamming distance between them. For example, rotating 45^o from one symbol to the next gives a SED=0.586, but Hamming distance d_H varies from 1 (between 000 at 0 rads and 001 at $\pi/4$ rads) to 3 (between 000 at 0 rads and 111 at $7\pi/4$ rads) for natural 8-PSK bit mapping. The distance between the all-zeros sequence and an error sequence is not representative of the distance between all correct and incorrect paths when the symbol mapping is not regular. Thus all sequences should be considered, not just the all-zeros sequence, recognizing that parallel branches of the differential trellis carry identical information symbols due to the rotationally invariant trellis; all parallel paths are equivalent.

Calculation of the minimum distance of concatenated systems with random interleavers is significantly more involved than considering the minimum distance of the component codes [3], [44], [45]. However, for the differentially-encoded turbo-coded modulation system with outer (3,2,2) parity code, the component codes are simple enough that determining the minimum distance error events of the concatenated system is possible. The input sequences to the inner decoder for this serially concatenated system are constrained to be interleaved codewords of the outer (3,2,2) parity-check code, which all have Hamming weight d_H=2, and thus the entire interleaved sequence is constrained to even Hamming weight. Any erroneous codeword sequence must also have even Hamming weight to be a possible decoded sequence. This ensures that a valid single two-branch error event will have even Hamming weight; otherwise the (3,2,2) parity decoder will change it to a sequence that does have even Hamming weight. Of course, multiple two-branch error events with individual odd Hamming weight can combine to form an error sequence with even total d_H, which would be a valid sequence. Clearly, however, these multiple two-branch error events will no longer be MSED error events. Multiple error events are not considered further in this chapter, but are examined in Chapter 8 with respect to interleaver design.

This system is expected to manifest a rather flat error floor due to the very low MSED of 0.586; simulation results presented later will demonstrate such an error floor. Lowering the error floor requires increasing the MSED or at least reducing the number of valid (*i.e.*, with total even Hamming weight) MSED/$d_{\min}$ two-branch error events.

The MSED error events are now considered in more detail, with the goal of better matching the 8-PSK mapping to the parity code sequence constraints to reduce the most likely error events. The minimum length detour of the inner decoder is six bits long for a two-branch error event. If natural 8-PSK mapping is used, the seven possible bit sequences for the two-branch error events with the all-zeros sequence as reference are given in the left side of Table 6.2.

Five out of seven of the naturally-mapped two-branch detours in Table 6.2 have even

8-PSK Natural Map						even
0	0	1	1	1	1	x
0	1	0	1	1	0	
0	1	1	1	0	1	x
1	0	0	1	0	0	x
1	0	1	0	1	1	x
1	1	0	0	1	0	
1	1	1	0	0	1	x

Improved 8-PSK Mapping						
0	0	1	$\rightarrow$	1	1	1
0	1	0	$\rightarrow$	0	0	1
0	1	1	$\rightarrow$	1	0	0
1	0	0	$\rightarrow$	0	1	0
1	0	1	$\rightarrow$	0	1	1
1	1	0	$\rightarrow$	1	1	0
1	1	1	$\rightarrow$	1	0	1

8-PSK Improved Map						even
1	1	1	1	0	1	
0	0	1	1	1	0	
1	0	0	0	1	1	
0	1	0	0	1	0	x
0	1	1	1	0	0	
1	1	0	0	0	1	
1	0	1	1	1	1	

Table 6.2: Bit sequences of two-branch error events for natural and improved 8-PSK mappings, compared to a reference correct all-zeros codeword sequence.

weight, and are permissible sequences. A mapping that maximizes the Hamming distance between mapped bits on adjacent symbols should further improve the concatenated code's distance spectrum. This mapping could be a type of anti-Gray, or maximal squared Euclidean weight (MSEW) mapping, as presented in [46]. Gray mapping minimizes the number of bit errors between adjacent symbols, or MSED error events. Both Gray mapping and MSEW mappings are discussed further in Section 6.1.2.

Such an improved mapping, presented in [47], is given in the center section of Table 6.2. The input bit sequences for the two-branch error events using the improved mapping with respect to the all-zeros sequence are shown on the right. Figure 6.1 shows this improved mapping, designed by Lance Pérez and his student Fan Jiang specifically for this (3,2,2)/D8-PSK SCC to provide a lower error floor by minimizing the number of valid MSED/$d_H = 2$ two-branch error events.

The improved mapping has only one even-weight sequence, 010-010, which generates a two-branch error event with a squared Euclidean distance (SED) of 4.0. The sequences 111-101 and 101-111 both have SED of 0.586, but are not eligible as two-branch error events because they are of odd weight. When all reference sequences are considered, not

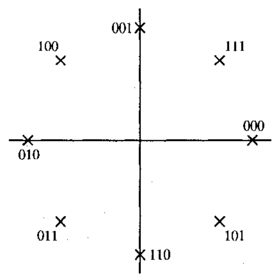


Figure 6.1: An alternative 8-PSK signal mapping with fewer MSED/$d_H = 2$ two-branch error events for the outer (3,2,2)/inner D8-PSK SCC.

just the all-zeros sequence, only one two-branch error event results in MSED=0.586 and minimum d_H=2. This MSED error event occurs with the interleaved coded sequence pair $\mathbf{v}', \hat{\mathbf{v}}'$=010-011, 011-010, and its seven parallel-path rotated versions, which are considered equivalent here and not counted. Exchanging $\mathbf{v}'$ and $\hat{\mathbf{v}}'$ also results in a $d_H = 2$ MSED two-branch error event; this exchanged version is considered equivalent to the original $\mathbf{v}', \hat{\mathbf{v}}'$ error event and also is not counted.

This improved mapping does have the same number of even-weight MSED two-branch error events, but far fewer (1 vs. 16) d_H=2 MSED error events in comparison to natural 8-PSK mapping. There are in total 128 possible MSED two-branch $\mathbf{v}', \hat{\mathbf{v}}'$ error events, if parallel paths are considered equivalent, and not counted. This is the same as assuming the initial trellis state of the diverging branch is zero, thereby eliminating all the rotated paths. Moreover, if only one of each error event pair $[\mathbf{v}', \hat{\mathbf{v}}'], [\mathbf{v}' = \hat{\mathbf{v}}', \hat{\mathbf{v}}' = \mathbf{v}']$ made possible by exchanging $\mathbf{v}'$ and $\hat{\mathbf{v}}'$ is counted, there are 64 total MSED two-branch error events possible with any 8-PSK mapping. All further analysis of MSED two-branch error events in this thesis considers rotated paths and exchanged error event pairs to be equivalent and does not count them.

Natural 8-PSK mapping has 16/64=1/4 of its MSED two-branch error events with $d_H = 2$, while the improved mapping has only 1/64 MSED two-branch error events with $d_H = 2$. This is a significant reduction in $d_{\min}$/MSED two-branch error events for the improved 8-PSK mapping.

Appendix A proves that a random interleaver without spreading is far more likely to contain a permutation allowing a d_H=2 MSED two-branch error event for both these mappings, with probability independent of interleaver length, than one for d_H=4 or 6, which decreases as $O(N^{-1})$. The improved mapping significantly reduces the number of d_H=2 MSED two-branch error events compared to natural mapping. This reduction of MSED multiplicity lowers the error floor as will be seen in Chapter 7. It does not increase the MSED of the code, which remains at 0.586 with high likelihood for random interleaving.

Use of a spread interleaver lowers the error floor further. An S-random interleaver [48], [49] is defined to have spreading S so that any two bits within a distance S of each other are interleaved by the S-random interleaver Π_S to a distance greater than S apart. In other words, for $|i-j| \leq S$, $|\Pi_S(i) - \Pi_S(j)| > S$. An S-random interleaver with spreading $S \geq 6$ will prevent the occurrence of a single two-branch error event by ensuring that only one bit per parity codeword is interleaved to adjacent 8-PSK symbols. Thus any error events would involve more than two branches, thereby doubling the MSED.

However, a spread interleaver cannot prevent the occurrence of multiple two-branch error events. The MSED of the concatenated code with a S-random spread interleaver of $S \geq 6$ is 1.172. S-random interleavers and the use of interleaver design to improve the error floor performance is discussed in greater detail in Chapter 8.

6.1.2 Other 8-PSK Bit Mappings

A few other 8-PSK mappings are also considered here. Gray mapping is a mapping designed so that adjacent 8-PSK symbols have minimum Hamming weight distance between them, which minimizes the number of bit errors which occur with a MSED symbol error. All neighboring symbols in the 8-PSK Gray mapping have $d_H = 1$ between them. Gray mapping is beneficial when no coding (or non-iterative decoding) is involved, and the receiver chooses its symbol estimate based on which PSK symbol is closest to the received noisy symbol. In this case, Gray mapping minimizes the errors associated at high SNR with the most likely MSED errors, and minimizes the BER.

However, as will be shown in the following section on EXIT analysis, Gray mapping performs very poorly in an iteratively-decoded system. This is precisely because all neighboring Gray-mapped symbols only differ by one bit. Iterative decoding provides *a priori* information on the bits from the other decoder. When iterative decoding is producing reliable *a priori* information, ideally the distance between symbols with only one bit difference would be maximized, and the MSED symbols would have maximum d_H whenever possible. This is because with reliable input *a priori* information, the APP decoder is much less likely to make an MSED error when more bits must be incorrect than with fewer bits required to be wrong.

All Gray-mapped MSED two-branch D8-PSK error events have $d_H = d_{\min} = 2$, with one bit error per branch. Thus Gray mapping is a very poor choice for this iteratively-decoded system, from a low error floor perspective as well as an iterative decoding perspective.

Two other 8-PSK mappings presented in the literature [46], [50] offer the distinct advantage of having no $d_H = d_{\min} = 2$ MSED two-branch D8-PSK error events. One 8-PSK mapping, the maximum squared Euclidean weight (MSEW) mapping [46], explicitly seeks to maximize the SED between symbol pairs with $d_H = 1$. This mapping seems ideally suited for this concatenated coding system. However, as simulations in Chapter 7 will show, the MSEW mapping does not perform as well in the error floor as either the improved mapping or the next mapping considered, the M8 mapping of [50] and [51]. Figure 6.2 shows the Gray, MSEW and M8 8-PSK bit mappings.

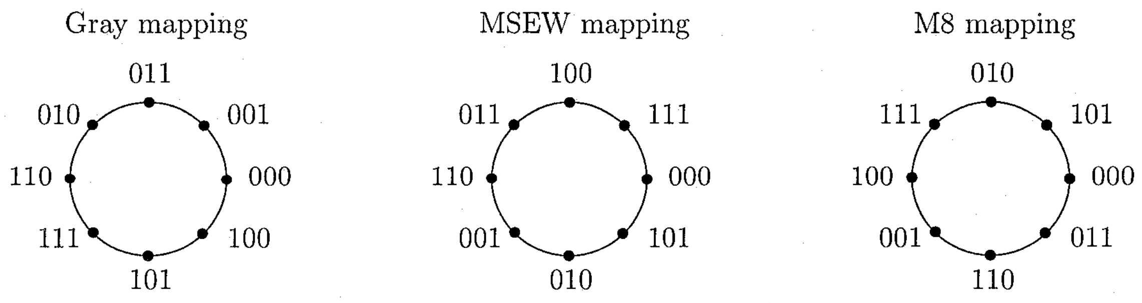


Figure 6.2: 8-PSK symbol constellation with Gray, MSEW [46] and M8 [50] bit mappings.

The M8 mapping was designed to optimize the total distance spectrum to minimize the pairwise error probability (PEP) $P(c \rightarrow e)$ of choosing the incorrect sequence $\mathbf{e}$ instead of the correct sequence $\mathbf{c}$ at high SNR. Simulations in Chapter 7 show that the M8 mapping provides the best high SNR performance, but only slightly better than the improved mapping, at the cost of 0.1 dB loss in turbo cliff. EXIT analysis in Section 6.2 confirms this loss in turbo cliff onset SNR.

6.2 EXIT Analysis

EXIT (EXtrinsic Information Transfer) analysis [4], [5] is a valuable technique for evaluating concatenated system performance in the turbo cliff, or waterfall, region. EXIT analysis uses mutual information as a measure of code performance. The concatenated code performance is simulated by EXIT analysis of the component codes, which is much simpler than simulating the entire concatenated system.

The mutual information $I(X;E) = I_E$ between symbols x and the extrinsic LLRs $E(x)$ is used as a measure of the reliability of extrinsic soft information E generated by each component decoder. Likewise, $I(X;A) = I_A$ measures the reliability of the *a priori* soft information A into the component decoder, with respect to x. Mutual information for binary symbols has a range of [0,1]. As the magnitude of A increases, the mutual information $I(A;x)$ between the LLR $A(x)$ and the symbol x it provides soft information about also increases.

Input *a priori* LLRs A are generated assuming a Gaussian distribution for $p(A)$, which has been shown to be a very good approximation, especially with increasing iterations [53]. Given A, with an associated $I(x;A) = I_A$, the component APP decoder will produce E, with associated $I(x;E) = I_E$. The inner decoder also requires channel metrics on the transmitted symbols, and thus is dependent on the channel SNR. The outer decoder of a serially concatenated system never sees the channel information and is independent of SNR.

In this manner, an EXIT chart displaying I_A, ranging from 0 to 1, versus I_E for each component APP decoder can be produced. Each component decoder is simulated individually, which is significantly faster than simulating the entire concatenated system. These component EXIT charts are used to study the convergence behavior of concatenated iteratively decoded systems. EXIT analysis becomes a useful tool for determining component code choice in a concatenated system as well as predicting system behavior in the waterfall or turbo cliff region, for large code lengths.

Mutual information is unchanged by the interleaving process; interleaving permutes the symbols but leaves the distribution unchanged. The interleaver also destroys any correlation between successive symbols. This separation and independence between the two decoders allows the component decoder EXIT charts to be combined into a single EXIT graph depicting the behavior of the iterative turbo decoding process.

6.2.1 Mutual Information Determination

Decoder performance may be evaluated by approximating the *a priori* input LLR distribution $p(A)$ to be Gaussian. This is a very good approximation with increasing iterations. The *a priori* LLR input A is modeled as a Gaussian random variable conditioned on the appropriate coded or uncoded symbol x,

$$A = \mu_A x + n_A, \quad n_A : \mathcal{N} : (0, \sigma_A^2) \tag{6.6}$$

where the mean $\mu_A = \sigma_A^2/2$. The noise n_A incorporates the effect of decoding uncertainty as well as the channel noise.

For a random variable A and BPSK transmitted symbols X, the mutual information $I(A;x) = I_A$ can be found as [54]

$$I_A = \frac{1}{2} \sum_{x=-1,1} \int_{-\infty}^{\infty} p_A(\xi|X=x) \log_2 \frac{2p_A(\xi|X=x)}{p_A(\xi|X=-1) + p_A(\xi|X=1)} d\xi, \tag{6.7}$$

where $p_A(\xi|X=x)$ may either be known theoretically or determined via simulation.

For A modeled as in Equation 6.6, Equation 6.7 becomes

$$I_A(\sigma_A) = \int_{-\infty}^{\infty} \frac{e^{-\frac{(\xi-\sigma_A^2/2)^2}{2\sigma_A^2}}}{\sqrt{2\pi}\sigma_A} \left(1 - \log_2\left[1 + e^{-\xi}\right]\right) d\xi, \tag{6.8}$$

as in [5]. Using Equation 6.8 for several σ_A values, each generating a vector of Gaussian-distributed A according to Equation 6.6, the corresponding $I_A(\sigma_A)$ values can be determined to obtain a set of $[\sigma_A^2, I_A(\sigma_A)]$ pairs ranging from approximately (0,1).

Alternatively, an approximation for Equation 6.8 expressing the relationship between σ_A and I_A as a piecewise polynomial in σ_A is given in [55] as

$$I(\sigma) \approx \begin{cases} -0.04211\sigma^3 + 0.20925\sigma^2 - 0.0064\sigma, & 0 \leq \sigma \leq 1.6363; \\ 1 - \exp(0.00181\sigma^3 - 0.01427\sigma^2 - 0.0822\sigma + 0.05496, & 1.6363 < \sigma < 10; \\ 1, & \sigma \geq 10. \end{cases} \tag{6.9}$$

An inverse approximation of $\sigma(I)$ is also given in [55].

EXIT analysis proceeds as follows:

1. For a chosen (σ_A^2, I_A) value, *a priori* LLRs A for an input sequence x are generated and fed into the APP decoder of interest. Specific details are given below for inner and outer decoders.

2. Given *a priori* LLR input A and channel metrics if needed, the APP decoder produces output extrinsic LLRs E.

3. The conditional probability density functions of $p(E|x = 1)$ and $p(E|x = -1)$ are calculated numerically from the output extrinsics E.

4. I_E is determined from numerical integration of Equation 6.7. Alternately, it can be calculated from $I_E = H(E) - H(E|x)$ where the entropy $H(E)$ and conditional entropy $H(E|x)$ are determined using the numerical simulations of the conditional pdf of E.

5. One point on the EXIT curve, corresponding to $[I_A, I_E]$, is obtained.

   Note: since these are numerical Monte Carlo simulations, their accuracy depends on having a sufficient number of random samples. There may be a slight deviation of I_A from its value according to (σ_A^2, I_A) depending on the number of samples of A. However, I_A may be recalculated, using the simulated values of A to obtain $p(A|x = 1), p(A|x = -1)$ and Equation 6.7 as per the calculation of I_E in the previous step. Typically the deviation of the numerically calculated I_A from the theoretical I_A of Equation 6.8 is minimal.

6. Choose another value of (σ_A^2, I_A) and begin again. Stop after computing I_E for $I_A \approx 1$.

6.2.2 Inner Decoder

For the inner decoder, noisy channel symbols $\mathbf{y}$ must also generated. The channel LLRs and inner *a priori* LLRs A_i serve as input to the inner decoder, which produces extrinsic LLRs E_i. The mutual information I_{E_i} is then calculated. In this manner, a plot of I_{A_i} vs. I_{E_i} may be generated which quantifies, for a given channel SNR, how the inner decoder performance improves with increasingly accurate *a priori* information.

However, the differential 8-PSK APP decoder operates on a non-binary symbol alphabet. The inital input LLRs must be converted to symbol probabilities, and output symbol probabilities converted back to LLRs, just as in the iterative decoding process, for the interface between the inner D8-PSK symbol decoder and outer binary/LLR decoder. Input bits $\mathbf{v}'$ are generated randomly and LLR values A_i calculated from those bits according to σ_A^2 corresponding to the desired $I(A; v')$. The input bits $\mathbf{v}'$ are mapped to 8-PSK symbols $\mathbf{w}$ and thence to differential 8-PSK symbols $\mathbf{x}$. The differential 8-PSK symbols are transmitted across an AWGN channel to arrive at the receiver as noisy symbols $\mathbf{y}$. Channel metrics are calculated according to the Gaussian conditional PDF of Equation 5.1 for each of the 8 possible 8-PSK symbols. Similarly, the *a priori* LLRs A are converted first to *a priori* bit probabilities $p_a(\mathbf{v}')$, then combined into symbol probabilities $p_a(\mathbf{w})$. The probability of a particular 8-PSK symbol is the product of its component bit probabilities, as per equation 5.6. These symbol probabilities are used as *a priori* input to the inner decoder.

Output symbol probabilities $p_e(\mathbf{w})$ are generated and converted to bit probabilities $p_e(\mathbf{v}')$ through marginalization, as per equation 5.2 for the concatenated system. Extrinsic LLRs E_i are calculated from $p_e(v')$, the conditional PDF $p(E_i|v')$ is determined numerically and $I(E_i; v')$ found for the inner code as in Equation 6.7. The EXIT chart for the inner code plots $I(A_i; v') = I_{A_i}$ vs $I(E_i; v') = I_{E_i}$ at a given channel SNR.

6.2.3 Outer Decoder

The outer code EXIT chart is determined in much the same manner; however, as the outer code does not receive channel LLR input, it is independent of channel SNR. Only the *a priori* LLR input A_o needs to be simulated. Since the parity code works with bits, the bit to symbol conversion used in the inner D8PSK decoder is unnecessary. Note that A_o is conditioned on $\mathbf{v}$, the (3,2,2) parity encoded bits, because the outer decoder receives soft information from the inner code on $\mathbf{v}'$, which is deinterleaved to become soft information on the coded bits $\mathbf{v}$. The information sequence $\mathbf{u}$ is generated randomly, encoded to $\mathbf{v}$, and LLR values A_o found according to σ_A^2 for $I(A_o; v) = I_{A_o}$. The parity decoder takes A_o as input, producing LLR output E_o. As for the inner decoder, the conditional PDF $p(E_o|v)$ is determined numerically and $I(E_o; v) = I_{E_o}$ calculated. The outer code EXIT chart plots $I_{A_o;v}$ vs I_{E_o}.

6.2.4 Combined EXIT Charts

As mentioned earlier, the EXIT charts for inner and outer decoder may be combined to determine system performance at a given SNR. The inner EXIT chart is positioned with I_{A_i} on the horizontal axis, and I_{E_i} on the vertical axis. The outer EXIT chart is positioned with axes swapped; I_{E_i} is on the horizontal, and $I_{A_i;v}$ is on the vertical. This allows visualization of the transfer of information between decoders with each iteration.

The system will operate within the channel created between the two component EXIT curves, bounded by the curves. The output I_E of one decoder will be the input I_A of the other decoder.

Initially, decoding begins with the inner decoding receiving uniform *a priori* LLRs, or $I_{A_i} = 0$. The corresponding I_{E_i} is generated from the output extrinsics and sent to the outer decoder as its *a priori* I_{A_o}. From these, the outer decoder produces I_{E_o} according to its EXIT curve, which is sent to the outer decoder as new *a priori* I_{A_i}. This completes one decoding iteration.

If the component EXIT curves cross over, closing off the channel, the decoders will be unable to improve on the extrinsic information being exchanged, and decoding will be stuck at a high bit error rate. This corresponds to the high BER region before the turbo cliff. At a certain SNR, the EXIT curves will just touch, barely blocking the channel. This is termed **pinchoff** and indicates the onset of the turbo cliff region. Increasing the SNR slightly beyond pinchoff provides a narrow channel to allow iterative decoding to converge to a very low error rate (indicated by $I_A = I_E = 1$). The narrow channel enforces only small increases in mutual information, indicating many iterations are necessary to achieve a low error rate at SNRs along the waterfall or turbo cliff region.

6.2.5 Outer (3,2,2) Parity Code and Inner D8-PSK Modulation

Figure 6.3 shows the EXIT chart for a differentially-encoded turbo-coded modulation system, using differential 8-PSK as the inner code, with I_{A_i} on the horizontal axis and I_{E_i} on the vertical axis, and a (3,2,2) parity check code as the outer code, with the axes swapped. The inner decoder information transfer curves are dependent on SNR$= \log_{10}(E_b/N_0)$ and are shown for SNR 3.4 and 3.6 dB. Natural 8-PSK mapping is indicated on the plot by 'nat map' and the improved mapping presented in Section 6.1.1 is designated by 'new map'.

Natural 8-PSK mapping allows for earlier turbo cliff onset at SNR 3.3 dB, but produces a higher error floor, as is shown in Chapter 7. Note that at SNR 3.4 dB, the

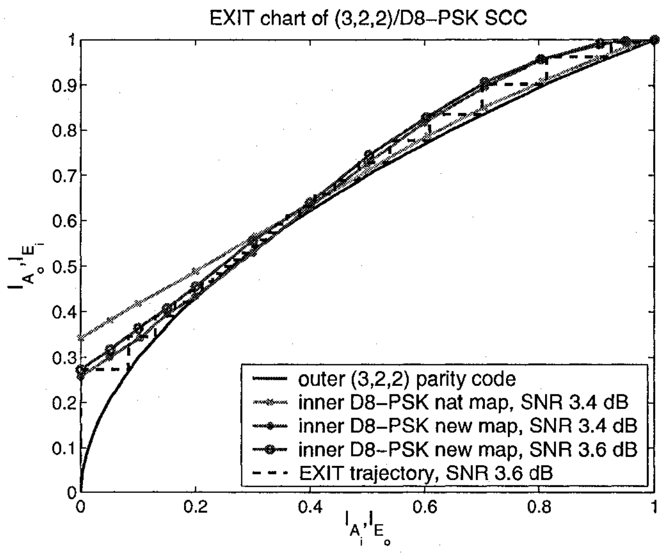


Figure 6.3: EXIT chart for outer (3,2,2) parity code and inner differential 8-PSK modulation, at SNR=$10\log_{10}(E_b/N_0)$=3.4 and 3.6 dB, with improved mapping; decoding trajectory shown for SNR=3.6 dB. Natural mapping of inner differential 8-PSK modulation is also shown for SNR 3.4 dB.

natural 8-PSK mapping EXIT curve provides an open channel for decoding to proceed to convergence, while the improved mapping is at pinchoff at SNR 3.4 dB.

An EXIT trajectory for the improved 8-PSK mapping at SNR 3.6 dB is displayed also, showing nicely how decoding follows the path delineated by the two component EXIT curves. Convergence occurs in about 25 iterations. Each vertical-horizontal step indicates one complete decoding iteration. The vertical lines are the increase in mutual information through the inner decoder, and the horizontal lines are the increase in mutual information obtained through the outer decoder. All curves are simulated for a 180,000 bit interleaver size.

The significant advantage of EXIT analysis is that the turbo decoder performance near the 'turbo cliff' region may be predicted without running simulations of the complete turbo decoder; EXIT transfer curves are obtained for each individual decoder.

Notice the close fit between the outer (3,2,2) parity check and inner differential 8-PSK EXIT curves at SNR 3.4 dB. The two codes are well-matched, in the sense that the combined codes minimize turbo cliff onset compared to a set of less well-matched codes.

6.2.6 Outer Convolutional Code and Inner D8-PSK Modulation

The outer (3,2,2) parity check code, while very well-matched to the inner differential 8-PSK modulation code in terms of turbo cliff onset SNR, is a weak code, with only four possible codewords and minimum Hamming distance $d_{\min} = 2$. A stronger outer code, with greater minimum distance, would probably provide better high SNR performance. Therefore, use of an outer convolutional code in place of the (3,2,2) parity check code is examined, to determine whether there is significant loss in turbo cliff onset SNR compared to the outer (3,2,2) parity code.

An EXIT curve using an outer rate 2/3 16-state maximal free distance recursive systematic convolutional code as the outer code instead of the (3,2,2) parity code is shown in Figure 6.4, together with an inner differential 8-PSK EXIT curve for SNR 5 dB. Natural mapping is used. The minimum distance $d_{\min}$ of this convolutional code is 5, compared to 2 for the (3,2,2) parity check code. The increase in $d_{\min}$ of the outer code increases the $d_{\min}$ of the concatenated code and lowers the predominant error floor found with the outer parity check code, as is shown in Chapter 7. However, reduction of the error floor comes at a significant increase of 1.6 dB in turbo cliff onset SNR.

As shown in Figure 6.4, pinchoff for the convolutional code and differential 8-PSK modulation occurs at SNR 5 dB, 1.6 dB past that of the concatenated system with the outer (3,2,2) parity code. It is clear from the EXIT curve of the outer rate 2/3 convolutional code that the differential EXIT curve must be raised significantly higher by increasing the SNR in order to clear the outer EXIT curve, and provide a channel for iterative convergence. This increase in turbo cliff onset is due to the poor match or fit, as shown by EXIT chart, between component decoders.

EXIT analysis has also proven quite useful in analyzing the potential of channel estimation based on the extrinsic information available from the D8-PSK APP decoder in the initial iteration. This topic is discussed in further detail in Chapter 9.

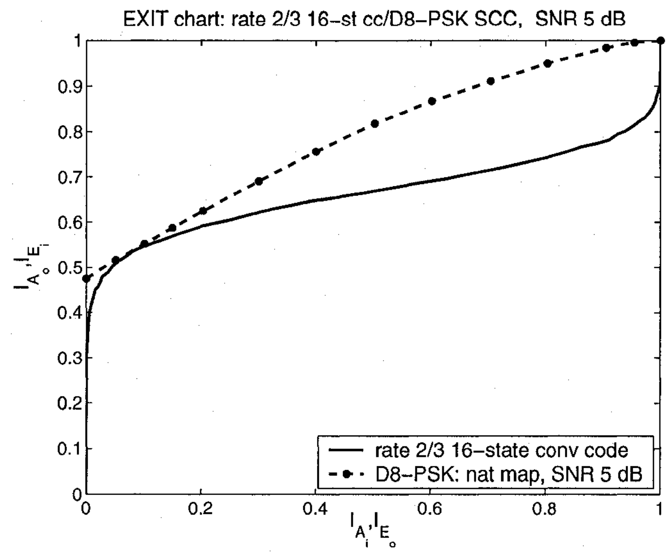


Figure 6.4: EXIT curve for outer rate 2/3 16-state recursive systematic convolutional code and differential 8-PSK at SNR=5 dB.

6.2.7 EXIT Analysis of 8-PSK Mappings

EXIT analysis is now applied to the results of Section 6.1.2 for various 8-PSK mappings. Both the MSEW mapping of [46] and the M8 mapping of [50] have no $d_{\min}$/MSED two-branch error events, and therefore these mappings are expected to have better high SNR performance. However, there may be some loss in turbo cliff onset SNR incurred with these 8-PSK mappings, compared with natural mapping and the improved mapping presented in Chapter 6.1.

Figure 6.5 displays a combined EXIT chart of the outer (3,2,2) parity check code and three inner differential 8-PSK mappings, for different SNR values. Gray mapping, MSEW mapping and M8 mapping are all considered. Gray mapping is expected to have the worst/highest error floor performance, as all MSED two-branch error events are also $d_{\min}$ events, but it also shows the highest turbo cliff onset SNR, above SNR 6 dB as shown in Figure 6.5.

This high turbo cliff onset SNR for Gray mapping is due to the fact that, although D8-PSK using Gray mapping provides the largest amount of extrinsic information of any

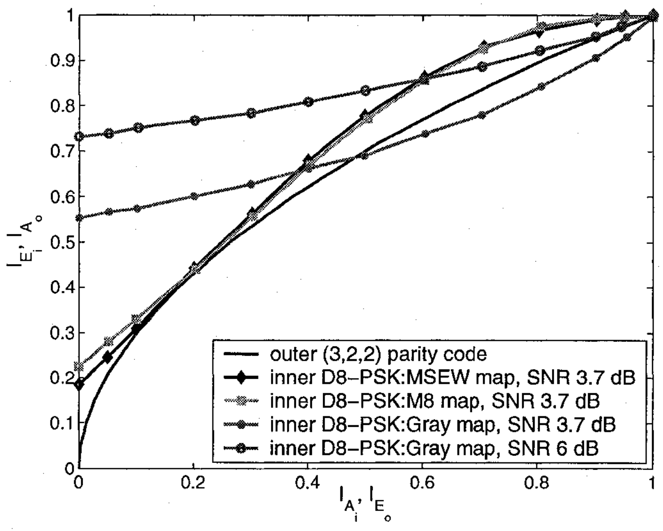


Figure 6.5: EXIT curve for outer (3,2,2) parity check code and various inner differential 8-PSK mappings.

of the mappings considered when no *a priori* information is available, it also shows little improvement in extrinsic information as the *a priori* information improves, providing the lowest amount of extrinsic information with good *a priori* information, $I_{A_i} > 0.5$. When Gray mapping is considered by itself as a modulation code, for example in iteratively-decoded bit-interleaved coded modulation (BICM-ID), EXIT analysis of Gray mapping shows there is no mutual information improvement with increasing *a priori* information [50], and thus Gray mapping is unsuitable as a modulation code for an iteratively-decoded system. When used as differential modulation, Gray mapping shows some modest improvement in mutual information as the quality of *a priori* information available to it improves. Still, the turbo cliff onset SNR of Gray-mapped D8-PSK is so high as to render the mapping ineffective for the modulation code in the differentially-encoded turbo-coded system.

The other interesting facet of the Gray-mapped EXIT curve is that pinchoff occurs at quite high values for *a priori* and extrinsic information. An open decoding channel exists at low mutual information values, but closes off when the curves touch at higher mutual information values. Decoding below but near pinchoff SNR will show some improvement in BER performance with iterative decoding, due to the open channel at lower mutual information values. However, decoding will be unable to converge to low BER until at SNRs above pinchoff.

The MSEW mapping and M8 mapping shown in Figure 6.5 both display pinchoff at SNR 3.7 dB, resulting in a 0.3 dB loss in turbo cliff onset compared to the improved 8-PSK mapping of Chapter 6.1. This loss in performance is costly, but these mappings are still practical as modulation codes.

In the following Chapter 7, simulation results for the BER performance of these different mappings, used in the inner differential 8-PSK modulation code, concatenated with the outer (3,2,2) parity code will be presented, along with BER results for the improved and natural 8-PSK mappings.

Chapter 7

Coherent Decoding Performance

7.1 Coherent Decoding

Coherent decoding assumes that perfect phase, carrier, timing and frame synchronization between transmitter and receiver have already been achieved before the received noisy symbol values are sampled and sent to the decoder. In the case of coherent decoding, the pulse modulation shape does not need to be considered, as all received values will be sampled at the correct time, which is when the modulation pulse for the symbol of interest is at its peak, and all other pulses are at a null. The issue of sampling at the correct time is discussed in greater detail in Chapter 10. Additionally, as phase and carrier synchronization have already been achieved, there is no phase offset to rotate the received noisy symbol.

The assumption of coherent decoding is reasonable in a circuit-switched system which has time to establish synchronization between transmitter and receiver, via a training sequence or pilot symbols, before transmitting encoded data. A phase-locked loop (PLL) and timing error detector (TED) can then be used prior to the decoder to track small phase and timing variations, and correct them.

For a packet-based system, coherent decoding may not be as good an assumption. The receiver does not know when packets will be arriving, and should have the ability to synchronize on the fly. Additionally, a mobile transmitter introduces significant channel variations which could be beyond the ability of the PLL to follow. An easily-implemented phase estimation technique that is integrated into the iterative decoding loop and utilizes the APP probabilities from the inner decoder is presented in Chapter 9. Similarly, a TED integrated into the iterative decoding loop is presented in Chapter 10. The two estimators are combined in Chapter 11 to form an integrated phase and timing synchronization/decoding combination.

This chapter considers coherent performance of the differentially-encoded turbo-coded modulation, assuming that phase and timing synchronization have already been achieved prior to decoding.

7.2 Outer Parity Code

Figure 7.1 shows the BER performance of the serially concatenated outer (3,2,2) parity code/inner differential 8-PSK system for both natural and improved 8-PSK mapping and random interleaving. Results are shown for interleaver sizes of 15000 bits and 180000 bits. Notice the lowered error floor of the improved mapping. Also shown is the improved mapping with a fixed S-random spread interleaver of S=9 and 15000 bits. The spread interleaver lowers the error floor further by doubling the code's MSED, as mentioned previously in Chapter 6.1. The effect of interleaver design on the error floor is discussed in greater detail in Chapter 8.

At a rate of 2 bits/symbol, 8-PSK capacity is at $E_b/N_0 = 2.9$ dB [8], [9]. The serially-concatenated (3,2,2)/D8-PSK system provides good BER performance at 0.6 (with naturally-mapped 8-PSK) and 0.8 dB (for the improved 8-PSK bit mapping) away from capacity for large interleaver sizes.

The BER performance of this concatenated system underscores the effectiveness of simple component codes, combined with analysis techniques, in designing and optimizing concatenated iteratively-decoded systems for excellent performance. Not only are the two component decoders very simple to implement (an 8-state trellis decoder for the inner code and a lookup table for the outer code) but taken individually, their error control potential is very limited. The (3,2,2) parity check code is very weak, with a minimal $d_H = 2$, and differential 8-PSK modulation alone is used for its independence from phase synchronization, rather than any error-correcting ability. Together, however, they unfold the full potential of turbo coding, outperforming even large 8-PSK Ungerböck trellis codes [23] by 1 dB. Uncoded QPSK reaches a BER of 10^{-5} at 9.5 dB; a 64-state 8-PSK trellis-coded modulation (TCM) code achieves a BER of 10^{-5} at 6.1 dB [21], for a coding gain of 3.4 dB at a rate of 2 bits/symbol. As shown in Figure 7.1, the differentially-encoded turbo-coded modulation system with a 180 kbit interleaver provides performance results 2.6 dB better than 64-state 8-PSK TCM at a BER=10^{-5}, for a coding gain of 6 dB with respect to uncoded QPSK for large interleaver sizes. An interleaver size of 15 kbit reduces the coding gain somewhat to 5.6 dB.

Along the turbo cliff, 50 decoding iterations are required for convergence. EXIT analysis predicted that a large number of iterations along the turbo cliff would be required for convergence, due to the well-matched EXIT curves of the component codes. A low SNR turbo cliff onset results, but only a narrow tunnel in the combined EXIT chart exists for any iterative improvement at near-turbo-cliff SNRs. A minimum of 50 frame errors per data point were collected along the waterfall; at higher error-floor SNRs, 25 frame errors per BER data point were observed.

As predicted by EXIT analysis, the larger interleaver size shows turbo cliff onset at SNR=3.3 dB for natural mapping and 3.5 dB for the improved mapping. Natural mapping provides a 0.2 dB advantage in turbo cliff onset, at the cost of a higher error floor. For the smaller interleaver size, along the turbo cliff, convergence requires 50 iterations; at SNR 4 dB, 30 iterations are required for convergence, decreasing to 10 iterations at SNR 5 dB and above.

The coherently-decoded differentially-encoded turbo coded modulation system using an outer (3,2,2) parity check code achieves a BER of 2×10^{-6} at an SNR of 3.9 dB for random interleaving, with an interleaver size of 15,000 bits using the improved 8-PSK

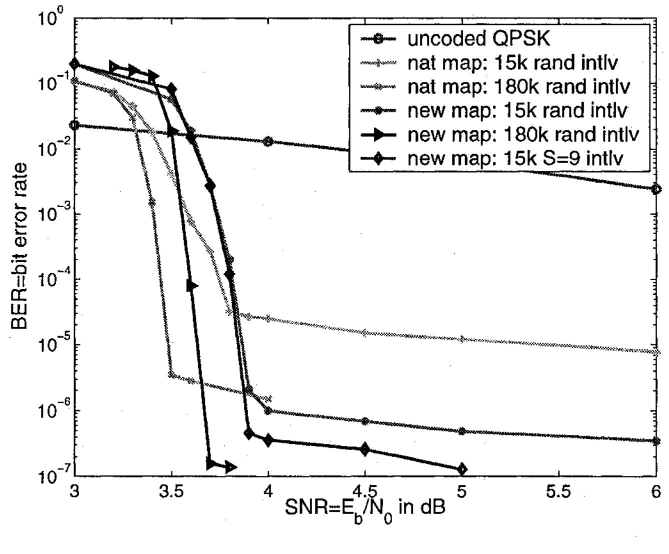


Figure 7.1: Performance of the serially concatenated D8-PSK system with outer (3,2,2) parity code over AWGN channel and coherent decoding, for various interleaver sizes, natural and improved 8-PSK mappings, with 50 decoding iterations.

mapping. In comparison, the serially-concatenated turbo trellis-coded modulation (SC-TTCM) system presented in [56] provides a BER of 2×10^{-5} at SNR=3.7 dB for coherent decoding of an outer rate 4/5 8-state convolutional code and inner 4-state rate 5/6 2×8PSK TCM and an interleaver size of 16,385 bits. The highest SNR value simulated in [56] was 3.7 dB. The system rate was 2 bits/symbol and 8-PSK modulation over an AWGN channel was used.

Bit-interleaved coded modulation using iterative decoding (BICM-ID) has been examined for 8-PSK modulation [51]. With an outer rate 2/3 16-state convolutional code and outer 8-PSK modulation code for a rate of 2 bits/symbol over an AWGN channel with coherent decoding, BICM-ID achieves a BER of 10^{-5} at SNR=4.5 dB with an interleaver size of 6000 bits, for a coding gain of about 5 dB.

The coherently-decoded D8-PSK/(3,2,2) parity-coded SCC presented in Figure 7.1 provides comparable performance to the SC-TTCM system, and superior performance to BICM-ID, and in addition, offers the potential for decoding without channel state information (CSI). System performance in the presence of phase offset, without CSI, will be examined in Chapter 9.

7.3 Different 8-PSK Bit Mappings

This system displays a flat error floor at high SNR, even with the improved 8-PSK mapping due to the existance of $d_H = 2$/MSED two-branch error events. As mentioned in Chapter 6.1.1, two 8-PSK mappings presented previously in the literature have no MSED $d_H = 2$ two-branch error events, and one, the MSEW mapping [46], maximizes the Hamming weight between MSED-separated symbols. These mappings are therefore expected to lower the error floor and slope. Gray mapping is expected to perform very poorly due to its many MSED $d_H = 2$ two-branch error events.

Figure 7.2 shows simulation results for the outer (3,2,2) parity/inner D8-PSK SCC over an AWGN channel, assuming coherent decoding, for the following 8-PSK bit mappings of $\mathbf{v}'$ to symbol $\mathbf{w}$: Gray mapping, MSEW mapping, M8 mapping [50], [51], and the improved mapping of Chapter 6.1.1 and [47]. Fifty decoding iterations were used, although along the error floor, convergence is obtained in 25 iterations or fewer. Random interleaving was used; a new random interleaver was generated for each codeword frame.

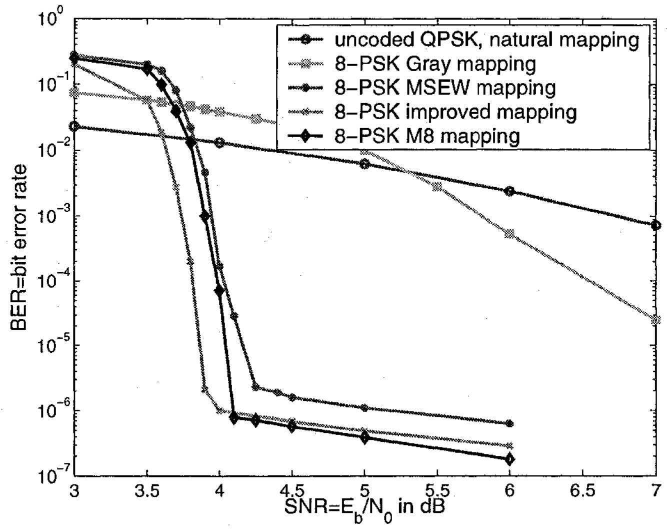


Figure 7.2: Performance of the serially concatenated D8-PSK system with outer (3,2,2) parity code for various 8-PSK bit mappings, random interleaving, AWGN channel and coherent decoding, 50 decoding iterations.

As expected from both distance and EXIT analysis, Gray mapping performs very poorly, and is unsuitable for use in an iteratively-decoded concatenated system. Both the MSEW and M8 mappings have a higher turbo cliff onset SNR, about 0.2 dB greater than the improved mapping. This was also predicted by EXIT analysis. Surprisingly, the MSEW mapping also has a higher error floor than the improved mapping despite maximizing the Hamming weight in two-branch MSED error events. The M8 mapping provides slightly better error floor performance than the improved mapping.

If the MSEW mapping maximizes the SED for $d_H = 2$ two-branch error events, and contains no MSED $d_H = 2$ two-branch error events, why then does it have worse error floor performance than the improved mapping? The answer is due to the fact that with random interleaving, the probability that a random interleaver instantiation contains at least one $d_H = 2$ two-branch error event coming from one (3,2,2) parity codeword is much greater than the probability that it contains at least one $d_H = 4$ two-branch error events, which must come from two (3,2,2) parity codewords, as shown in Appendix A. Note that if these MSED error events deinterleave to single bits in different (3,2,2) parity codewords, the outer parity decoder will, with high probability, correct those single bits.

Thus if no $d_H = 2$ MSED two-branch error events exist, the primary decoding error mechanism now occurs with $d_H = 2$/SED=2 two-branch error events (the next smallest SED after MSED=0.5816) rather than $d_H = 4$ MSED two-branch error events. These SED=2 error events occur with an actual channel MSED symbol error rather than a SED=2 symbol error from the channel. However, in these decoding errors, the D8-PSK trellis stabilizes at a SED=2 symbol error which causes only two bit errors ($d_H = 2$), both deinterleaving to the same (3,2,2) parity codeword. It does this in preference to the MSED symbol error which a hard decision of the channel symbols would provide, as the MSED symbol error would cause four bit errors ($d_H = 4$) at a minimum. The decoder will most often correct these four bit errors unless they deinterleave to two (3,2,2) parity codewords.

An example of this is found in Figure 7.3, which shows an actual MSEW-mapped error event. The correct sequence is the solid bold path, the incorrect decoded $d_H = 2$

SED=2 sequence is the dashed bold path, and the dotted path shows the $d_H = 4$ MSED sequence which would have resulted from a hard decision decoding of the received D8-PSK channel symbol. The bits mapping to each 8-PSK symbol associated with the branches are shown, with incorrect bits shown in bold.

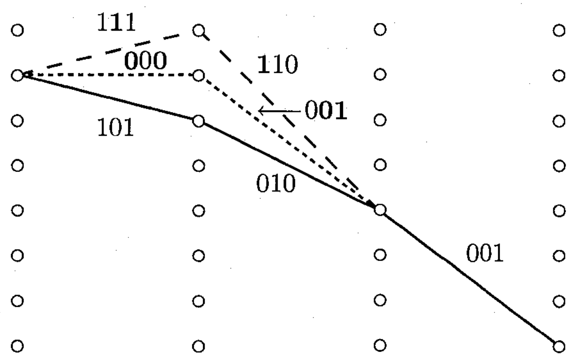


Figure 7.3: A $d_H = 2$ SED=2 two-branch error event for the outer (3,2,2) parity/inner differential 8-PSK SCC with MSEW bit mapping.

The MSEW mapping has 64 possible $d_H = 2$ SED=2 two-branch error events, which far outweighs the single $d_H = 2$ MSED and 9 $d_H = 2$ SED=2 two-branch error events for the improved mapping of Chapter 6.1.1. As earlier, parallel events are not counted, so the initial state of the diverging branch is assumed zero; also, exchanging $\mathbf{v}'$ and $\hat{\mathbf{v}}'$ is not counted. The M8 mapping has only 16 $d_H = 2$ SED=2 two-branch error events and no $d_H = 2$ MSED two-branch error events.

Adjustment of the 8-PSK mapping to decrease $d_H = 2$/MSED error events provides only partial mitigation of this error floor. To decrease the error floor further, the interleaver design must be considered. A code-matched interleaver design which eliminates the MSED error events of this code is presented in the following Chapter 8.

7.4 Outer Convolutional Code

Simulation results for a differentially-encoded turbo coded modulated system using an outer rate 2/3 16-state maximal free distance recursive systematic convolutional code and inner differential 8-PSK encoding with natural mapping are presented in Figure 7.4. Coherent decoding is assumed, and the transmission channel is AWGN with circularly-symmetric zero-mean Gaussian noise. This serially-concatenated code has the same rate of 2 bits/symbol as the previously-presented SCC using an outer (3,2,2) parity code, with which it is compared to in Figure 7.4. Random interleaving of 15,000 bits was used for each codeword frame. Uncoded QPSK performance, with a rate of 2 bits/symbol, is also presented for reference.

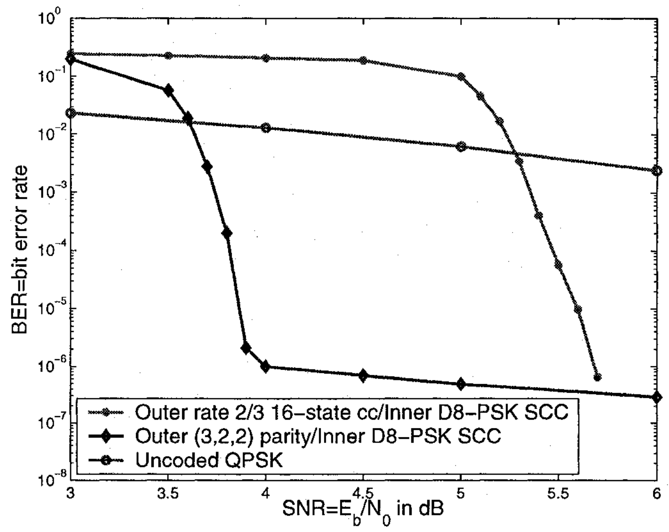


Figure 7.4: BER vs SNR performance over AWGN channel of the serially concatenated D8-PSK system with outer rate 2/3 16-state maximal free distance recursive systematic convolutional code and inner differential 8-PSK encoding using natural mapping, compared to outer (3,2,2) parity code concatenated with inner differential 8-PSK encoding using improved mapping.

Note that this outer convolutional-encoded system provides a lower error floor than the parity-encoded system, at the cost of a significantly increased waterfall onset SNR. By SNR 5.8 dB, the outer convolutional-encoded system gives better BER performance, but with a 1.6 dB loss in waterfall onset SNR, as predicted with EXIT analysis. The convolutional-encoded system is hardly a capacity-approaching design, and though providing better high SNR performance, is a poor compromise in terms of both approaching capacity and providing good error floor performance. The remaining portion of this thesis considers only the outer parity-encoded system, and will look at lowering the error floor further through appropriate interleaver design.

Chapter 8

Code-Matched Interleaver Design

8.1 Introduction

As shown in Chapter 7, the (3,2,2)/D8-PSK SCC developed in this thesis provides near-capacity BER performance. However, at higher SNR, the performance displays a significant error floor, caused by low-Hamming-distance, low-MSED error events. These low-MSED error events were initially discussed in Chapter 6.1.

This error floor is often seen in concatenated coding systems such as turbo codes; recall the performance of the original turbo code in Figure 4.1. The free-distance or minimum-distance asymptote of a concatenated code with low MSED is relatively flat, and at high SNR, the concatenated code's performance approaches the free-distance asymptote [44]. This error floor can be severe enough that a simple convolutional code or TCM code actually achieves better performance than the concatenated code at high SNR.

Considerable effort has gone into designing interleavers which improve the minimum distance of a concatenated code, thus improving its error floor. The classic approach is the S-random spread-interleaver design [48], [49]. The S-random interleaver spreads nearby bits far apart, breaking up low-weight error patterns so the interleaved coded output sequence has increased weight. The spreading constraint interleaves bits within a distance S to a distance greater than S apart.

Dithered relative prime (DRP) interleavers [57] are a recent design technique providing very large minimum distance. An input dither vector locally permutes the input sequence, a relative prime (RP) interleaver then spreads the dithered sequence, and an output dither vector locally permutes the interleaved sequence again. These techniques, however, while increasing minimum distance, do not directly seek to eliminate low-weight error events specific to a particular concatenated code.

Designing interleavers matched to the distance spectrum of a specific code, to eliminate specific low-weight input sequences producing low-weight coded sequences, is an extension of the spreading concept. Code matched interleavers (CMI) are presented in [58], [59] for PCC binary turbo codes with component recursive systematic convolutional (RSC) codes. Interleaver design constraints in addition to spreading are proposed to ensure that all bits spaced k_1 multiples of a distance μ apart are not interleaved to k_2 multiples of μ apart, where μ is the length of the shortest weight-2 input sequence producing a finite weight codeword. Limits are placed on k_1, k_2 depending on the maximum weight to be eliminated. A related constraint was designed for weight 4 codewords. Results for a rate 1/3 turbo code with component 4-state convolutional codes, interleaver size 4096 and spreading 35 and show a factor of 40 drop in the error floor at SNR 1.2 dB compared to random interleaving and a factor of 6 error floor drop compared to S-random interleaving with S=42.

The ST-random interleaver [60] has a similar constraint that, in addition to the spreading constraint, input symbols spaced T or less multiples of L apart are not interleaved to T or less multiples of L apart, where L is the shortest length of any weight-2 input sequence producing a finite length codeword, that is, the generator feedback polynomial $g_1(D)$ has $1 + D^L$ as a divisor. This is equivalent to the definition for μ previously. Performance results for a rate 1/2 turbo code show a drop in the error floor by a factor of 5 at SNR=2 dB for length 10000 interleaver and memory 3 component convolutional codes. The ST_s-random interleaver [61] extends the spreading constraint S to input symbols spaced $T_s \neq T$ or less multiples of L apart, and considers PR4 and EPR4 partial response channels. A slight drop in error floor by a factor of 2 is seen for a convolutional code serially concatenated with the channel precoder and interatively decoded, using a $(S, T_s) = (30, 10)$ interleaver compared with an S-random interleaver with S=30.

A code-matched interleaver design is examined for the intersymbol iterference (ISI) channel [62], with a punctured convolutional code and precoder viewed as a SCC. An S-random interleaver is constructed, tested for all weight-2 error events and altered until no weight-2 error event patterns appear. Results show an improvement of 0.3-0.4 dB at BER=10^{-6} over an S-random interleaver.

S-random interleaver design is extended to eliminate multiple low-weight error events with constraints for two and three error events given in [63]. However, the designed interleavers were unable to completely satisfy the given constraints, which were then relaxed until an interleaver could be found. Results were given for symbol-interleaved parallel concatenated trellis coded modulation (PCTCM) and show an improvement of 0.2 dB at BER=10^{-5} over random interleaving for 8-PSK PCTCM. Interpolated results for BER=10^{-6} show a 0.35 dB improvement.

Interleaver constraints are developed in this thesis for the (3,2,2)/D8-PSK SCC, to eliminate the low-squared-Euclidean-distance (SED)/low-weight error events of this code. These interleaver constraints eliminate specific multiple error events as well as single error events. Factor graph representation [64], [65] of the interleaver allows these multiple error events to be viewed as short cycles in a graph.

Interleavers which avoid low-weight error events for turbo codes have been derived from graphs with large girth [66]; the girth of a graph is defined as the length of its shortest cycle. The graph is designed to prevent two bits spaced short multiples of $P = 2^{\nu} - 1$ from interleaving to another short multiple of P apart, where ν is the memory of the component convolutional codes; these events would otherwise cause low-weight error events. The graph design also prevents two groups of two bits, where the two bits within a group are spaced short multiples of P apart, from interleaving to two new groups, with one bit from each of the de-interleaved groups going to each of the interleaved groups, where each bit within an interleaved group is spaced a short multiple of P apart.

This is similar to the ST and ST_s interleavers, but the construction technique is based on graphs. A rate 1/3 turbo code's minimum distance $d_{\text{min}} = 30$ using one of these designed interleavers of length 5049, and component codes of memory $\nu = 3$, while a random interleaver construction for turbo codes of the same interleaver length and code memory resulted in an average $d_{\text{min}} = 14$ and best $d_{\text{min}} = 22$ for the random interleavers [67].

A different interleaver design for serially-concatenated convolutional codes (SCCCs) is presented in [68] which similarly prevent two bits, or groups of bits, spaced short distances apart from interleaving to new groups spaced short distances apart. These interleaving connections are defined as in the previous work as short cycles, and interleavers that exclude these cycles up to and including length l are analyzed and designed. A $(\Delta, \gamma)_l$ interleaver is defined as excluding all cycles of length l where the distances between bits of each (de-interleaved) group are all less than Δ, and the sum of all distances between inter-group bits is less than γ. For memory rate 1/2 turbo codes with interleaver length 2700 and simple $\nu = 1$ component codes, a $(3, 9)_4$ interleaver has guaranteed $d_{\text{min}} \geq 5$, compared with a random interleaver with $d_{\text{min}} \geq 2$ and an $(s, t) = (20, 40)$ interleaver with $d_{\text{min}} \geq 3$. The (s, t) interleaver is similar to the S-random interleaver, except with non-symmetric separation factors; for the S-random interleaver, $s = t = S$, and for the (s, t) interleaver, $s \neq t$. Performance results with these same interleavers showed a drop in the error floor of two orders of magnitude for the $(3, 9)_4$ interleaver compared with the random interleaver, and an error floor drop of a factor of 7 compared to the $(20, 40)$ interleaver.

Interleaver construction rules which eliminate short cycles for a particular graph representation of the (3,2,2) parity/D8-PSK interleaver will be shown to also eliminate the lowest SED error events for this SCC. As the component codes for the SCC considered in this thesis are not convolutional codes, the constraints for preventing error events are somewhat different than those designed previously for turbo codes and SCCCs. A graph representation that examines parity codeword/8-PSK symbol bit connections will clearly show two-branch error events as short cycles.

8.2 Minimum Distance Error Events

The fully-connected differential trellis ensures that for every branch diverging from the correct path, there is another branch at the next time step merging back to the correct path. Thus the differential code has many possible two-branch error events, some with MSED=0.586, just as for 8-PSK modulation.

The MSED error event with respect to the all-zeros sequence is shown in Figure 8.1A. The incorrect differential symbols are $\hat{\mathbf{x}} = e^{j\pi/4}$, e^{j0}, with the diverging branch giving a squared Euclidean distance (SED)=0.586. The associated incorrect information symbols are $\hat{\mathbf{w}} = e^{j\pi/4}$, $e^{-j\pi/4}$ mapping to the bit sequence $\mathbf{v}'$=111-101. The error sequence **e**=111-101 has odd Hamming weight and is disallowed by the parity-check code. Incorrect bits are in bold.

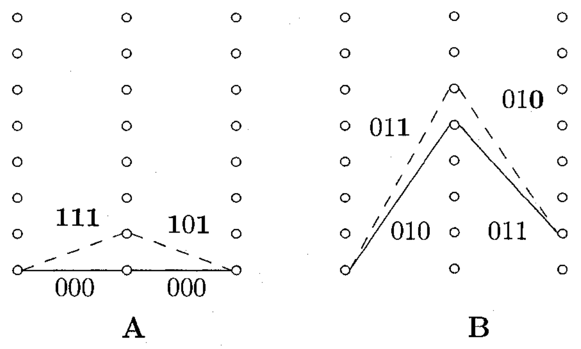


Figure 8.1: Minimum squared Euclidean distance (MSED) error events for the outer (3,2,2) parity/inner differential 8-PSK SCC with the improved mapping of Figure 6.1.

However, the 8-PSK mapping is not regular, as shown in Table 8.1 listing the SED vs. Hamming distance d_H between each 8-PSK symbol of this mapping. Rotating from one symbol to the next gives SED=0.586, but Hamming distance d_H varies from 1 to 3. Only

MSED	Hamming distance d_H=1
0.586	010/011
2.00	000/001,111/101,010/110
3.14	000/100,111/110,001/011,001/101
4.00	000/010,111/011,100/101
MSED	Hamming distance d_H=2
0.586	000/101,111/001,001/100,100/010,011/110,110/101
2.00	000/110,111/100,001/010,011/101
3.14	000/011,111/010
4.00	-
MSED	Hamming distance $d_H = 3$
0.586	000/111
2.00	100/011
3.14	010/101
4.00	001/110

Table 8.1: Squared Euclidean Distance (SED) vs. Hamming distance between all 8-PSK symbol mappings for the improved mapping of Figuree 6.1.

one symbol pair has SED=0.586 and d_H=1: 010/011. From this, the minimum distance MSED/min d_H two-branch error event is constructed as shown in Figure 8.1B. The correct sequence $\mathbf{v}'$ is 010-011, the diverging sequence $\hat{\mathbf{v}}'$ is 011-010, and error sequence $\mathbf{e}$=001-001 with d_H=2. This MSED two-branch error event is termed $E_{d_{\min}}$.

Not only does this sequence have even Hamming weight, so it can be a valid codeword depending on interleaving, but it also has the minimum possible Hamming weight for the parity sequence! The errors from this MSED/min d_H two-branch error event must come from a single (3,2,2) parity-check codeword, as each parity codeword has d_H=2. Due to the rotationally invariant differential trellis, all correct/diverging path sets parallel to this one are equivalent, with MSED=0.586 and d_H=2. Switching diverging and correct paths obtains the same MSED/d_H.

As seen from Table 8.1, there are many possible error events with MSED but larger Hamming distance. The next lowest weight two-branch error events with MSED either have d_H=2 per branch or one branch with $d_H = 1$ and one branch with $d_H = 3$, for total d_H=4. These error events are termed E_{dH4}. Two possible correct/diverging E_{dH4} path sets are shown in Figure 8.2, with 8.2A showing $\mathbf{v}'$=111-110, $\hat{\mathbf{v}}'$=001-011, and $\mathbf{e}$=110-101, for SED=0.586 and d_H=4. A $d_H = 3/d_H = 1$ weight-4 MSED two-branch error event is shown in Figure 8.2B, for $\mathbf{v}'$=000-011, $\hat{\mathbf{v}}'$=111-010, and $\mathbf{e}$=111-001. Incorrect bits are in bold.

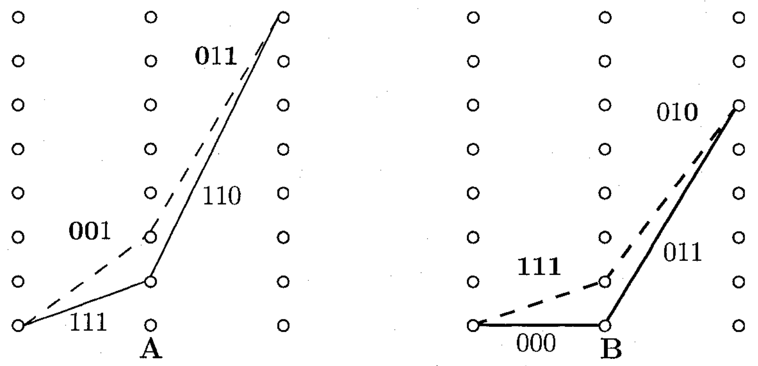


Figure 8.2: Minimum squared Euclidean distance (MSED) E_{dH4} error events with $d_H = 4$ for the outer (3,2,2) parity/inner differential 8-PSK SCC, using the improved mapping of Figure 6.1.

Another low-weight error event consists of the minimum distance error event $E_{d_{\min}}$ with an intermediate parallel branch between diverging and merging branches, to make it a three-branch error event as shown in Figure 8.3. Here the correct sequence $\mathbf{v}'$=011-001-010, $\hat{\mathbf{v}}'$=010-001-011, and $\mathbf{e}$=001-000-001. No extra Hamming distance accumulates, but the parallel branch is another MSED detour, so the SED doubles to 1.14. This type of error event is termed E_{par}. Additional parallel branches increase the SED by 0.586 for each branch.

Finally, consider an error event consisting of two separate $E_{d_{\min}}$ error events, as shown in Figure 8.4. This error event has SED=1.14 and d_H=4. Three separate $E_{d_{\min}}$ error events gives SED=1.76 and d_H=6. In general, L $E_{d_{\min}}$ error events provide SED=L*0.586 and d_H=2L.

The following Section explores how interleaver design may eliminate these low-weight error events.

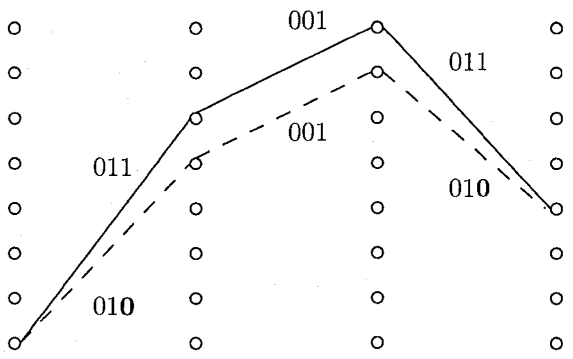


Figure 8.3: A three-branch error event E_{par} for the outer (3,2,2) parity/inner differential 8-PSK serially concatenated SCC.

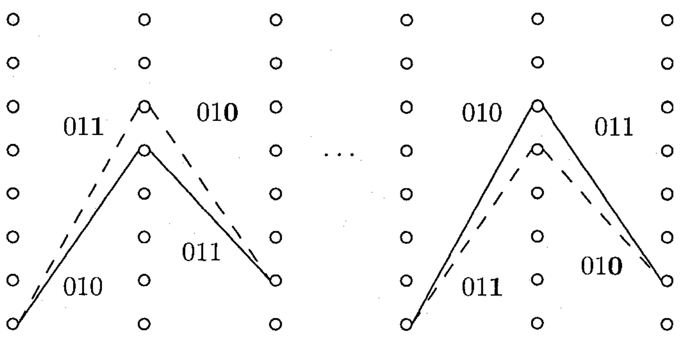


Figure 8.4: Two MSED error events for the outer (3,2,2) parity-check/inner differential 8-PSK SCC.

8.3 Interleaver Design

The interleaver interface $\Pi(\mathbf{v})$ between the (3,2,2) parity codeword sequence $\mathbf{v}$ and the interleaved bits $\mathbf{v}'$ mapped to an 8-PSK symbol sequence $\mathbf{w}$ may be represented as a factor graph. Figure 8.5A shows this graph representation. Circles represent the bits $\mathbf{v}$ of the (3,2,2) parity-encoded sequence, squares represent the interleaved bits $\mathbf{v}'$, and edges connecting circles to squares represent the interleaver bit permutations $\Pi(\mathbf{v})=\mathbf{v}'$.

The MSED/min d_H two-branch error event $E_{d_{\min}}$ is depicted in Figure 8.5A. Only the erroneous bit edges or connections are shown. If bits of each (3,2,2) parity codeword are combined into a larger codeword node (ovals), and bits mapped to each 8-PSK symbol into a larger symbol node (rectangles), the graph of Figure 8.5B is obtained. Each codeword node has three edges for its three component bits, and each 8-PSK symbol node similarly has three edges. This codeword/symbol graph representation clearly shows that only one parity codeword and two adjacent 8-PSK symbols are involved in $E_{d_{\min}}$.

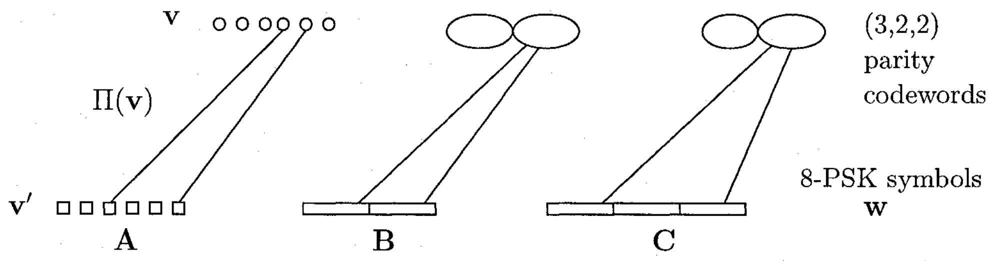


Figure 8.5: Interleaver factor graph for MSED two-branch and three-branch error events.

Figure 8.5C shows the three-branch d_H=2 error event E_{par} in codeword/symbol graph representation. This error event also only involves one parity codeword and two 8-PSK symbols; this time, the two erroneous 8-PSK symbols are not adjacent but separated by the parallel branch symbol.

A spread bit interleaver Π of length N eliminates error events $E_{d_{\min}}$ and E_{par} with the following simple constraint which spreads bits from the same codeword a distance S apart:

$$\text{if } 1 \leq \ |i-j| \ < 3, \text{then } |\Pi(i) - \Pi(j)| \geq S, \ \forall\, i,j = 1,\ldots,N; \ i \neq j. \qquad (8.1)$$

To eliminate the error event $E_{d_{\min}}$, the spreading S must be ≥ 6; to eliminate the error event E_{par}, the spreading S must be ≥ 9. The two-branch error event E_{dH4} is also eliminated with spreading $S \geq 6$, as two erroneous parity codewords connect to two adjacent 8-PSK symbols.

This spreading constraint is more restrictive than needed, as it applies as well to adjoining bits from two codewords, which is unnecessary. For a codeword node I with bits i_1, i_2, i_3, $V(I) = [i_1, i_2, i_3]$ is the set of bits in codeword I. $\Pi(V(I))$ are the interleaved bits which $V(I)$ maps to, and $d(\Pi(V(I))_{k,l}) = |\Pi(i_k) - \Pi(i_l)|$ is the distance between any two bits of codeword I. The codeword spreading constraint becomes

$$d(\Pi(V(I))_{k,l}) \geq S,\ \forall k,l = 1,2,3; k \neq l;\ \forall I = 1,\ldots,N/3 \qquad (8.2)$$

Spreading eliminates single error events, but not multiple error events. Thus the minimum distance for this SCC with a spread interleaver of $S \geq 6$ remains at MSED=1.17.

The next low-weight error event consists of two $E_{d_{\min}}$ error events spaced apart as shown in codeword/symbol graph representation in Figure 8.6. Two parity codewords and four 8-PSK symbols participate in this error event. Considering two adjacent 8-PSK symbols as a single larger (dashed) node reveals this event to be a 4-cycle loop. Notice that both the parity codewords and the two two-symbol nodes are an arbitrary distance apart; hence spreading cannot eliminate this pattern. However, note that both codewords go to the same two adjacent-symbol nodes. Transplanting one edge of this 4-cycle loop to another non-adjacent symbol breaks the loop and eliminate the error event. Spreading is assumed sufficient to ensure that only one bit/edge per codeword permutes to any given 8-PSK symbol or its directly adjacent symbols.

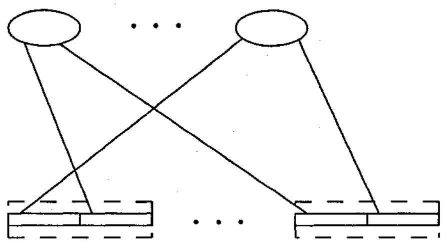


Figure 8.6: Interleaver codeword/symbol factor graph for two MSED two-branch error events; 4-cycle loop.

The following additional constraint is then placed on an S-random interleaver: no two parity codewords may have more than one bit each permuted to the same two adjacent-symbol nodes. For the set of bits $V(I)$ in codeword I mapping to bits $\Pi(V(I))$ in symbols $S(I)$, $X(I)_1$ is the set of 8-PSK symbols adjacent to $S(I)$. For now, **adjacent** is considered to mean directly adjacent. The **neighborhood** of I at level k is termed

$X(I)_k$; $X(I)_1$ contains the symbols adjacent to the first level of I, that is, the directly permuted bits of I, $\Pi(V(I))$. Deinterleaving the bits of the neighborhood $X(I)_1$, gives a set of codewords $V(I)_2$, the codewords connected to the bits of codeword I through the adjacent neighborhood $X(I)_1$. If the same codeword J appears twice in the set $V(I)_2$, a 4-cycle loop is formed between that codeword and codeword I, as in Figure 8.6. In other words, two permuted bits from parity codeword I have adjacent symbols which deinterleave back to the same parity codeword J, forming a 4-cycle loop.

Elimination of the 4-cycle loop may be implemented in the interleaver with the constraint on bits $V(I) = [i_1, i_2, i_3]$ following the notation above.

4-Cycle-Free Interleaver Constraint:

$$V(i_i)_2 \cap V(i_j)_2 = \emptyset, \text{for } i, j = 1, 2, 3;\ i \neq j \tag{8.3}$$

In a similar fashion, three two-branch error events are shown in Figure 8.7. Three parity codewords and six 8-PSK symbols, or three two-symbol nodes, are connected in this error event. Use of the dashed adjacent two-symbol node representation shows this event to be a 6-cycle loop.

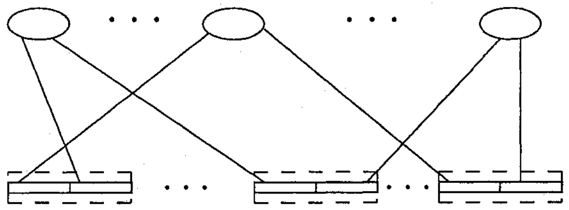


Figure 8.7: Interleaver codeword/symbol factor graph for three MSED two-branch error events; 6-cycle loop.

To eliminate the 6-cycle loop, the definition of neighborhood must be extended to include one more level. If the bits of $V(I)_2$ not directly connected to $X(I)_1$ are interleaved to bits in symbols $S(I)_2$, then $X(I)_2$ is the set of 8-PSK symbols adjacent to $S(I)_2$. This is the second level of adjacent neighborhood. Again, deinterleaving the bits of the neighborhood $X(I)_2$ gives a set of codewords $V(I)_3$. If the same codewords J and K appear in both $V(I)_3$ and $V(I)_2$, a 6-cycle loop is formed. The following constraint eliminates this 6-cycle loop.

6-Cycle-Free Interleaver Constraint:

$$V(I)_2 \cap V(I)_3 = \emptyset \tag{8.4}$$

Higher multiples of two-branch error events, generating larger cycles, may be considered. Each additional error event generates a cycle increase of 2. Elimination of this larger cycle requires an extension of the current adjacent neighborhood to one more level, where there must be no overlap with symbols in lower levels.

Additionally, multiples of three-branch or higher error events can be considered. These require an extension of the adjacency definition of the neighborhood to account for the symbols in the middle branches which are not in error. Two three-branch events with a parallel central branch require an extension of adjacency to the two neighboring symbols on either side. Elimination of 4-branch events with two parallel central branches require adjacency of three neighboring symbols on either side.

Table 8.2 lists the expected MSED for random, S-random, 4-cycle-free, 6-cycle-free and 8-cycle-free interleavers, with required spreading and adjacent neighborhood. The **adjacent neighborhood** is the number of 8-PSK symbols to either side of the 8-PSK symbol of interest that must be considered. Different adjacent neighborhoods are required for eliminating different cycle lengths. For example, the 6-cycle-free interleaver must have both 6-cycles and 4-cycles eliminated; to eliminate 4-cycles that would contribute an MSED of 1.758, an adjacent neighborhood of 2 must be used, while eliminating 6-cycles with MSED=1.758 requires adjacency of only 1.

Interleaver	MSED	Adjacent Neighborhood	Spreading
Random	0.586	-	- - -
S-random	1.174	-	$S \geq 6$
4-cycle-free	1.758	4-cycle: 1	$S \geq 9$
6-cycle-free	2.344	6-cycle: 1, 4-cycle: 2	$S \geq 12$
8-cycle-free	2.930	8-cycle: 1, 6-cycle: 2, 4-cycle: 3	$S \geq 15$

Table 8.2: MSED and required adjacent neighborhood and spreading for various interleavers with (3,2,2) parity/differential 8-PSK SCC.

8.4 Simulation Results

Interleavers were constructed according to the constraints given in Section 8.3. All cycle-free interleavers applied spreading to bits of one codeword. The 4-cycle-free interleavers were constructed with an adjacent neighborhood of 1, and 6-cycle free interleavers with an adjacent neighborhood of 2 applied to the 4-cycle constraint. Spreading for each codeword of the 4-cycle-free interleavers was 10, while for 6-cycle-free, S=15. An S-random interleaver with S=9 was also constructed. Each interleaver was 15,000 bits long.

BER simulations were run for the outer (3,2,2) parity-check/inner differential 8-PSK SCC using random, S-random, 4-cycle-free and 6-cycle-free interleavers. Simulation results are shown in Figure 8.8.

Use of a spread interleaver with $S = 15$, which doubles the minimum distance over random interleaving, lowers the error floor by a factor of about 5. The 4-cycle-free interleaver triples the minimum distance w.r.t. random interleaving and provides a significant improvement in performance, with a factor of 20 lower error floor. The MSED for the 6-cycle-free interleaver, 2.344, is quadrupled in comparison to that of random interleaving, and shows a factor of 50 improvement in the error floor performance.

Note that the error floor slope change with increasing d^2_{min} is not dramatic. There is a slight increase in error floor slope as the interleaver choice increases the MSED. At high SNR, the probability of bit error asymptotically approaches the BER due to MSED codewords alone. The high-SNR error floor is thus well-approximated by Equation 6.5.

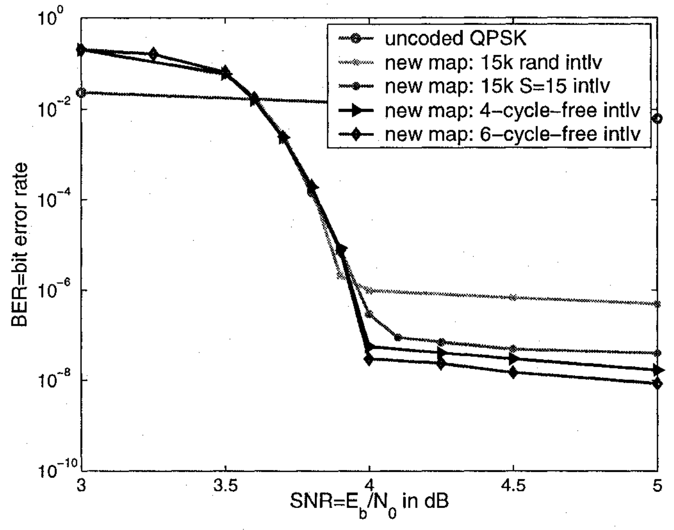


Figure 8.8: BER vs SNR for (3,2,2)/D8PSK SCC with random, S-random and short-cycle-free interleavers.

Figure 8.9 shows the slope of several MSED BER asymptotes, using Equation 6.5 with $E_s = RE_b$, for SNR ranging from 4 to 5 dB. Their average multiplicity, $A_{d^2_{\min}}$, is normalized to 1, as we are interested in seeing the relative slope between each asymptote as $d^2_{\min}$ increases, as well as any drop in floor when the multiplicity stays the same but $d^2_{\min}$ increases.

Comparing the error floor slope between adjacent $d^2_{\min}$ asymptotes, we see that the change in error floor slope with each increase in MSED is visible but slight. Thus we do not expect to see a dramatic increase in error floor slope by increasing the MSED. Notice there is significant drop in the error floor as $d^2_{\min}$ increases. Doubling $d^2_{\min}$ from 0.586 to 1.172 results in a factor of 5 improvement in BER, and quadrupling the MSED to 2.344 results in a factor of 25-30 improvement. This improvement is exclusive of any improvement due to reduced multiplicity, as the BER was calculated with normalized multiplicity. This improvement in error floor asymptote between $d^2_{\min} = 0.586$ and $d^2_{\min} = 2.344$ is similar to the factor of 50 improvement in BER seen between random interleaving, with $d^2_{\min} = 0.586$, and a 6-cycle-free interleaver, with $d^2_{\min} = 2.344$.

Performance simulations in the remaining chapters of this thesis use a minimal spread-

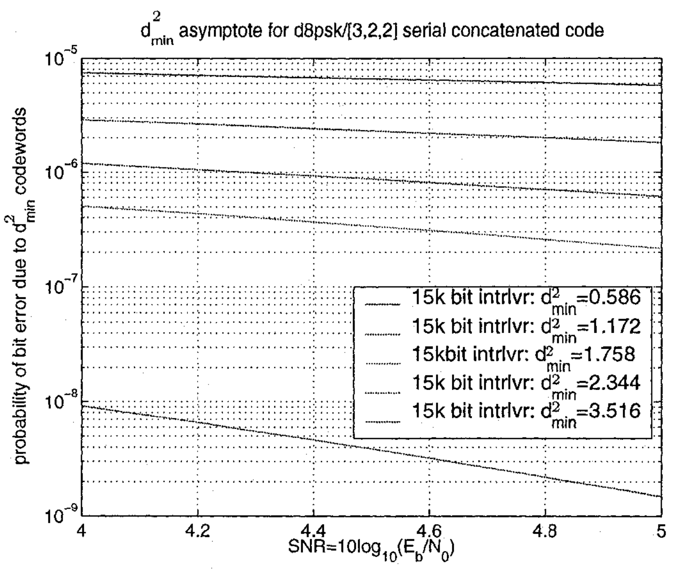


Figure 8.9: Probability of bit error due to MSED codewords ($d^2_{\min}$ asymptote) vs SNR for (3,2,2)/D8PSK SCC with increasing $d^2_{\min}$, normalized average multiplicity $A_{d^2_{\min}}$.

ing $S = 3$ 15 kbit S-random interleaver. This spreading ensures that only one bit per codeword is mapped to each symbol; however, it does not eliminate two-branch error events, as that requires $S = 6$, and thus the MSED for the $S = 3$ interleaver remains at 0.586 as for random interleaving. Use of the 4- or 6-cycle-free code-matched interleavers developed in this chapter in place of the $S = 3$ spread interleaver should lower the error floor in the simulations of Chapters 9, 10 and 11 by a factor of 20 or more, where coherent decoding results are achieved.

Chapter 9

Decoding Without Channel Information

9.1 Phase Synchronization

Phase synchronization is considered as a separate problem from timing synchronization in this chapter. Accurate timing estimation is assumed for the analysis and simulations of phase estimation that follow. Chapter 10 considers timing estimation and Chapter 11 examines combining phase and timing estimation together in the iterative decoding process. For the purpose of this Chapter, timing or symbol synchronization between the transmitter and receiver is assumed to be already achieved prior to the decoding process, and thus only phase synchronization integration into the decoding process is considered in this process.

At the low SNR values achievable with iteratively-decoded codes, issues such as phase synchronization become critical, especially for higher order modulations. Conventional phase synchronization utilizes a PLL (phase-locked loop) or Costas loop [69], resulting in phase ambiguities for PSK constellations. In addition, the squaring loss for higher order PSK modulation becomes significant. For 8-PSK suppressed-carrier signalling, the squaring loss of an eighth-power-law device at E_s/N_0=9 dB is upper-bounded by

10 dB [70] with respect to PLL operation on an unmodulated carrier. Typical PLL loop SNRs must be at least 6 dB to achieve synchronization [71], [72]; thus the eighth-power squaring device should see a minimum loop SNR of 16 dB to ensure achieving lock. High loop SNR requires narrow loop bandwidth, but loop bandwidth is inversely proportional to the PLL acquisition time [73]. Thus narrow loop bandwidth increases acquisition time, and is incompatible with the fast tracking and acquisition needed in wireless packet-messaging systems. Effective, fast phase synchronization for iteratively decoded systems using higher order PSK modulation becomes highly problematic.

A training sequence of known symbols may be sent initially to characterize any channel offset and allow synchronization between transmitter and receiver. For continued tracking and synchronization, known pilot symbols may be sent within the codeword frame, spaced every so often. Both these methods aid synchronization at the cost of lowered rate, as energy must be spent to transmit these synchronization symbols which do not convey any data. Neither training sequences or pilot symbols will be used for APP channel estimation, developed in Section 9.4.

The classical method of eliminating phase synchronization is differential M-PSK encoding with differential demodulation; however, a 3 dB loss in SNR vs BER occurs for M-PSK, with $M > 2$ [74]. This also applies to turbo coding. Differential BPSK modulation resulted in a 2.7 dB loss in SNR [75] for a rate 1/2 turbo code using differential demodulation. Such significant loss is counter to the near-capacity performance expected of turbo codes.

Various techniques to mitigate the penalty incurred by differential demodulation have been used. Multiple symbol differential detection [76], [77], [78] has been applied to iteratively-decoded serially concatenated codes with differential modulation [79], using linear prediction to obtain channel estimates. This technique results in an exponential expansion of the decoding trellis.

Another method presented in [80] that extends the phase memory to N symbols also results in an exponentially-increased decoding trellis, but lowers the complexity below

M^N states by using non-coherent sequence detection (NSD) with a soft-output Viterbi algorithm (SOVA) that allows trellis branch pruning to reduce the state complexity. This method, when applied to a serially-concatenated rate 1/2 16-state convolutional code with DBPSK modulation, shows 0.3 dB loss compared with coherent decoding at a BER of 10^{-5} using the modified BCJR decoding with $N = 4$. Similarly reduced-state (but still exponential in N) complexity sequence detection algorithms are presented in [81] with inserted pilot symbols as well as per-survivor-processing (PSP) of SOVA to reduce state complexity. For a rate 1/2 4-state outer convolutional code serially concatenated with QPSK modulation and pilot insertion rate of 1 pilot/16 coded symbols, the PSP-based receiver incurred 0.4 dB loss compared to perfect CSI at a BER of 10^{-5}.

An expanded-state decoding trellis is also presented in [82] for serial concatenation of a rate 1/2 convolutional code with differential M-PSK modulation. The channel phase is discretized into N states, resulting in a linear trellis expansion of MN states.

Similarly, in [83] iterative decoding of turbo codes with QPSK modulation incorporates channel estimation for fading channels by using quantized phase in an expanded "supertrellis". A simpler model which quadruples the size of the decoding trellis by discretizing the phase into one of four quadrants is presented in [84]. Performance improvement of 0.1 dB over an unmodified APP trellis is observed for a rate 1/3 turbo code using BPSK modulation with a Gaussian-distributed phase process and phase error variance $\sigma_{\varphi}^2 = 0.3$.

Instead of expanding the already complex APP decoding trellis, channel estimation of PSAM (pilot-symbol-assisted modulation) BPSK turbo codes over fading channels in [85] is accomplished in the iterative decoding block but outside the APP decoder to still allow for iteratively improving channel estimates. BER performance of 10^{-4} within 0.5 dB of coherent decoding for slow fading is achieved.

Turbo-embedded estimation (TEE) is an alternate approach that has been investigated for BPSK in [86] and extended to QPSK and 8-PSK [87]. This technique uses the most probable state during the forward recursion of the APP decoder to obtain a symbol decision, which is fed to a simple tracking loop to compute an updated phase estimate. No state expansion of the APP trellis, and corresponding increase in complexity, occurs. However, TEE requires an initial phase estimate to begin decoding; this phase estimate is obtained from a known preamble whose length is approximately 1.5% of the total packet length. Results over the AWGN channel, with phase noise according to a recent DVB-S2 proposal [88], for a rate 1/3 unpunctured turbo code using 8-PSK modulation with a system rate of 1 bit/symbol were 0.3 dB from coherent decoding at a BER of 10^{-5}, increasing with the punctured rate 1/2 turbo code using 8-PSK modulation with a system rate of 1.5 bits/symbol to 0.7 dB from coherent decoding at a BER of 2×10^{-3}.

Other channel estimation techniques incorporating estimation into the iterative decoding loop, but not within the APP decoding trellis, have also been developed. Factor graph models of iterative decoding with concatenated codes can be used to derive decoding algorithms which pass probability messages along the connecting edges of the factor graph [89], [90]. In this manner, some new estimation methods use factor graph representation of the joint *a posteriori* distribution of the received symbols, information symbols and the channel parameters to derive a framework for channel or phase estimation. The *a posteriori* symbol probabilities from the iterative decoder are input messages in the estimation factor graph. Models for block, Rayleigh and multi-path fading channels are presented in [91]. Message-passing algorithms for joint decoding and phase estimation on a factor graph for constant phase and random walk phase models are developed in [92] and [93].

Factor-graph modelling of a phase offset, assuming a Tikhonov distribution for the phase offset, is presented in [94]. Near-coherent performance is achieved for a rate 2/3 LDPC of length 64800 bits, mapped to 8-PSK symbols, with 32 pilot symbols used after every 1476 data symbols, over an AWGN channel with DVB-S2-compliant European Space Agency (ESA) phase noise model [95]. A constant frequency offset is added in [96] and estimated by discretizing to L levels. For a rate 1/2 (3,6) length 4000 LDPC mapped onto QPSK modulation, with 1 pilot symbol every 20 symbols, near-coherent performance is achieved with a frequency offset uniformly distributed from $[-\nu_0, \nu_0]$ with $\nu_0 T = 5 \times 10^{-3}$ for symbol timing T and discretized to $L = 3$ levels. The phase noise model is the same as in [94].

A simple carrier phase recovery method utilizing the extrinsic soft information in the iterative decoding loop of a turbo decoder is presented in [97]. This method uses an initial carrier phase acquisition algorithm to locate the phase offset quadrant, and a phase tracking algorithm to fine-tune the phase offset estimate. BPSK modulation is used. The acquisition algorithm uses N decoding iterations to test each of four possible multiples of $\pi/2$ phase offsets, and chooses the offset that maximizes the sum of the extrinsic LLR magnitudes, with a complexity of $4N$ decoding iterations to choose the phase quadrant. The tracking algorithm uses a feedback form based on the difference between the extrinsic LLR magnitude sum from one frame to the next.

An approximation to the ML log-likelihood function for the phase estimate that iteratively uses the APP soft information from the component APP decoders of a turbo decoder is presented in [98]. An approximation to the log likelihood function (LLF) for carrier phase estimation is found by expanding the LLF into a Fourier series with two harmonics for each symbol and summed over the entire block. The ML phase estimate maximizes the LLF, so the LLF derivative is zero when evaluated at the ML phase estimate. The APP soft information is used to average out the effect of the symbols in the LLF. Simulation results show coherent decoding results for a constant phase offset of 15^o for BPSK modulation of a rate 1/2 turbo-coded system. A similar procedure is used in [99] for a turbo-encoded bit-interleaved coded-modulation (BICM) system. A rate 1/3 turbo encoder composed of two 16-state convolutional encoders and a length 1024 interleaver are combined with 8-PSK modulation, for a rate 1 system. Near-coherent results are achieved with a constant phase offset of 10^o, increasing to 0.3 dB loss at a BER of 10^{-4} for a phase offset of 20^o.

9.2 APP Channel Estimation

If the inner differential 8-PSK APP decoder can provide reliable output APP soft information without any prior phase knowledge, even in the presence of a channel phase offset or channel response $\mathbf{h} = \exp(j\varphi)$, this APP soft information could be used to estimate the channel phase and iteratively improve the original input channel metrics $p(\mathbf{y}|\mathbf{x})$ to $p(\mathbf{y}|\mathbf{x}, \tilde{\mathbf{h}})$. Through iterations, decoding performance close to the complete phase knowledge scenario is achieved with integrated APP channel phase estimation. This method uses APP soft information from the differential decoder without expanding trellis complexity. Neither external phase estimation, such as a training sequence preamble or non-data-aided (NDA) phase estimation, nor differential demodulation are needed to begin the decoding process.

The iteratively improving extrinsic information available from the D8-PSK APP decoder is utilized in this chapter to estimate and track the phase through successive iterations, leading to a decoder with the ability to achieve convergence even in the presence of significant channel phase offset. This channel estimator is incorporated within the iterative decoding block but not within the trellis structure of the APP decoder itself, as that greatly increases decoding complexity.

The channel estimation method presented in Section 9.4 uses the D8-PSK APP soft information to form soft symbol estimates, used together with the received samples $\mathbf{y}$ to form a channel estimate. We term this method **APP channel estimation**. This method was presented originally in [100], [101], together with differential demodulation, in the serial concatenation of a convolutional code with a differential space-time code. Initial non-uniform *a priori* information on the space-time symbols was provided by differential demodulation for input to the inner APP decoder. Sufficient output extrinsic information allowed formation of a channel estimate to improve the channel metrics for the next iteration, along with *a priori* information from the outer APP decoder. Differential demodulation was only used for the initial iteration.

As will be shown in Section 9.3, initial differential demodulation is not necessary for channel estimation with the serially-concatenated code presented in this thesis. The differential inner code provides sufficient extrinsic information with APP decoding to allow iterative decoding to begin, even with significant channel phase offset.

Results from this thesis incorporating APP channel estimation into the iterative decoding process, without use of differential demodulation or pilot symbols, are presented in [6], [7] and [47].

A very similar phase estimation method for turbo-coded 16-QAM (quadrature amplitude modulation) was presented [102] at nearly the same time as [47], using iterative soft-decision-directed phase estimation based on approximate ML estimation for QAM. This method shows 0.2 dB loss at an approximate BER of 10^{-5} for a constant phase offset of 20^o, compared to coherent decoding. Extension to 4-QAM/QPSK in [103] showed approximately the same performance for the same phase offset. This algorithm is functionally identical to APP channel estimation. Both algorithms can be viewed as approximations to the iterative expectation-maximization (EM) algorithm [104], [105], as shown in [106]. The EM algorithm iterates by alternately finding the expectation of a likelihood function based on current parameter estimates and received data, and obtaining new parameter estimates from the maximized likelihood function. The EM approach was originally applied to uncoded carrier phase recovery in [107] using hard symbol decisions for joint phase estimation and data detection, or averaging over soft symbol estimates from an APP decoder for non-data-aided (NDA) phase estimation.

As mentioned in Chapter 5, the differential 8-PSK trellis is rotationally invariant to phase shifts of multiples of $\pi/4$ rads or 45^o. Thus, the estimated phase offset only has to be accurate to mod $(\pi/4)$ rads. Since the APP soft information $p(\mathbf{x}|\mathbf{y})$ of the channel symbols $\mathbf{x}$ is already provided by the inner APP decoder for each iteration, that soft information can be used to provide a channel estimate. The question is whether the soft information $p(\mathbf{x}|\mathbf{y})$ in the initial iteration is sufficiently reliable to begin the estimation process. Note that the soft information is not required to be extremely reliable in the sense of providing an immediately accurate phase estimate. It merely needs to provide enough information to allow an iteratively-improving phase estimate. EXIT analysis is an excellent tool for examining this question.

9.3 EXIT Analysis of Inner Code with Phase Offset

An EXIT chart of the differential 8-PSK decoder operating with various phase offsets is shown in Figure 9.1 for an AWGN channel at SNR=4.5 dB. The phase offset Φ is constant over the entire symbol block; 180,000 bits were used for simulating each EXIT point. The coherent decoding case of perfect phase synchronization, 0^o phase offset, is shown in the topmost curve. The constant channel phase offset is $\exp(j\Phi)$ and the received noisy symbols are $\mathbf{y} = \mathbf{x}\exp(j\Phi) + \mathbf{n}$.

Neither differential detection nor pilot symbols were used. The D8-PSK APP decoder clearly provides some extrinsic information even initially, when no *a priori* information is available. This is shown by the fact that when $I_A = 0$, $I_E \geq 0.2$, even for the worst case phase offset of $\varphi = \pi/8$ radians (halfway between two symbols).

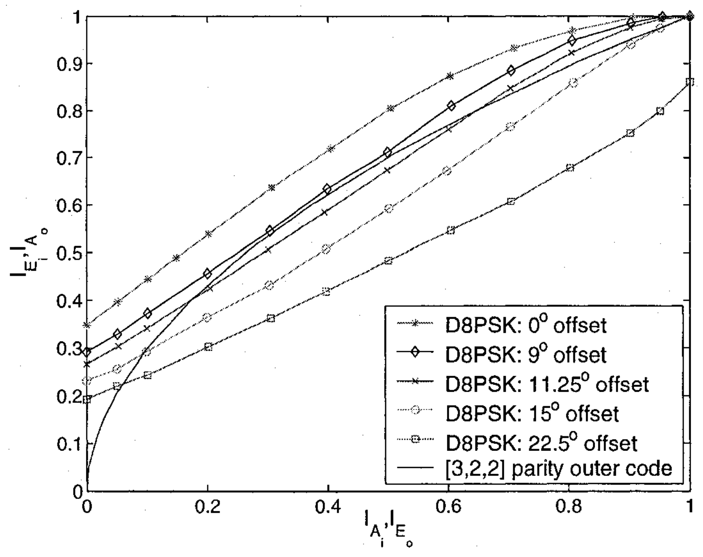


Figure 9.1: EXIT chart of the differential 8-PSK APP decoder using the new mapping, for various constant phase offsets, SNR=4.5 dB, and (3,2,2) parity check decoder.

The presence of extrinsic information without any *a priori* information is very significant. No external method prior to the decoder of generating initial phase information, such as a training sequence or non-data-aided (NDA) phase estimator, will be required to provide an initial phase estimate, nor is differential demodulation needed.

The initial extrinsic information from the D8-PSK APP decoder allows decoding to begin in the presence of a phase offset, but is insufficient to complete convergence at the displayed SNR. The component EXIT curves intersect in Figure 9.1, and decoding stops at the intersection point, corresponding to unreliable decoding and a high BER. However, this extrinsic information can be used to form an initial channel phase estimate, giving better input channel metrics for the following decoding iteration, which in turn, allows the decoder to improve the extrinsic information in the next iteration beyond what is possible without the better channel metric. Improved extrinsic information results in an even better phase estimate. As the channel phase estimation improves with iterations, the differential 8PSK EXIT curve with phase offset approaches the coherent curve, allowing for convergence as the phase estimate reaches the correct channel offset.

The inner APP decoder generates both 8-PSK symbol probabilities $p(\mathbf{w})$ and D8-PSK symbol probabilities $p(\mathbf{x})$; the latter will be used as input to a channel estimator for subsequent iterations. Low complexity is important, given the emphasis on implementation simplicity leading to our choice of component codes. The channel estimator complexity must not overshadow that of the decoding system, and thus an optimal linear estimator such as a minimum mean square error (MMSE) estimator is not considered. A simpler filtering estimator is used, as presented in [6], [7], [47], and [101].

9.4 Channel Estimation

The channel model used herein is AWGN with complex noise variance N_0 and a complex time-varying channel gain $\mathbf{h}$. In the case of unknown channel phase, $\mathbf{h}$ is a unit-length time-varying rotation. A time-varying phase offset can change with every symbol. The received signal $\mathbf{y}$ is given as

$$y_k = h_k x_k + n_k; \quad h_k = e^{j\varphi_k},\ n_k : \mathcal{N}(0, \sigma_n^2), \tag{9.1}$$

where h_k is the instantaneous complex time-varying channel gain at symbol time k.

Taking the first moment, or expectation E, of Equation 9.1, gives

$$E[h_k] = y_k / E[x_k]. \tag{9.2}$$

The expectation of each symbol x_k, $E[x_k]$, is taken over the *a posteriori* probabilities $p(x_k|y_k)$ from the inner D8-PSK APP decoder, using them as *a priori* $p(x_k)$ according to the definition of expectation:

$$E[x_k] = \sum_{m=0}^{7} \exp\left(\frac{j2\pi m}{8}\right) p\left(x_k = \exp\left(\frac{j2\pi m}{8}\right)\right). \tag{9.3}$$

Normalizing Equation 9.3 to unit amplitude gives the instantaneous expected value, or soft symbol value, for x_k as

$$\tilde{x}_k = \frac{E[x_k]}{|E[x_k]|}. \tag{9.4}$$

Note that while $E[\mathbf{x}] = 0$, $E[x_k] \neq 0$ provided that the APP extrinsic probabilities $p(x_k)$ are not uniform. The differential code provides non-uniform extrinsic information for all SNRs of interest, i.e. in the turbo cliff region and above, as shown by EXIT analysis.

An instantaneous channel estimate is then found from Equation 9.2 as

$$\hat{h}_k = y_k \tilde{x}_k^*. \tag{9.5}$$

The instantaneous channel estimates $\hat{h}_k$ are first normalized as

$$\hat{h}_{k,\text{norm}} = \frac{\hat{h}_k}{\left(\sum_{k=1}^{N} |\hat{h}_k| / N\right)}. \tag{9.6}$$

Then the instantaneous channel estimates $\hat{h}_{k,\text{norm}}$ are filtered through a lowpass filter F to provide smoothed filtered estimates $\tilde{h}_k$. The lowpass filter F is chosen based on the expected phase process, described in greater detail in Section 9.5, which considers three different phase offset models. A simpler filter is appropriate for a constant phase offset, although the more complex filter used for time-varying phase processes will work for a constant phase offset as well but is more complicated than is required.

New complex channel metrics for the D8-PSK APP decoder in the next iteration are calculated using the filtered channel estimates $\tilde{h}_k$, as

$$p(y_k|x_k, \tilde{h}_k) = (\pi N_0)^{-1} \exp(-|y_k - \tilde{h}_k x_k|^2/N_0) \tag{9.7}$$

Each iteration improves the APP values of $p(\mathbf{x}|\mathbf{y})$ from the inner APP decoder, and an improved channel estimate $\tilde{\mathbf{h}}$ is calculated from the latest APP values every iteration.

As the *a posteriori* probabilities $p(\mathbf{x}|\mathbf{y})$ are used to form the normalized channel estimate $\tilde{\mathbf{h}}$, we term this procedure **APP channel estimation** or APP phase estimation. Figure 9.2 shows a block diagram of the iterative decoding process with the APP phase estimation block enclosed in the dashed rectangle.

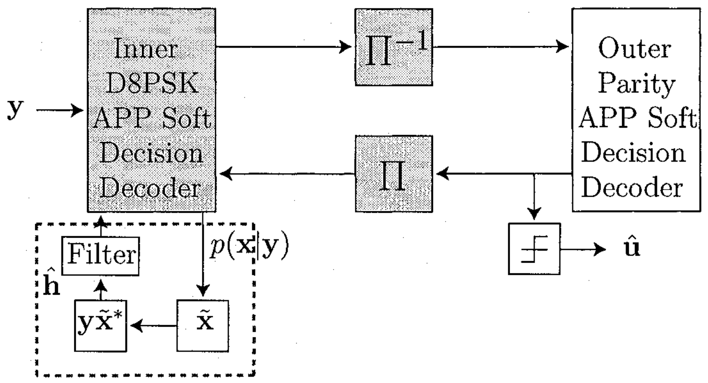

y
Inner D8PSK APP Soft Decision Decoder
Π^{-1}
Π
Outer Parity APP Soft Decision Decoder
$\hat{\mathbf{u}}$
Filter
$\hat{\mathbf{h}}$
$p(\mathbf{x}|\mathbf{y})$
$\mathbf{y}\tilde{\mathbf{x}}^*$
$\tilde{\mathbf{x}}$

Figure 9.2: Serial turbo decoder, D8-PSK inner APP decoder and (3,2,2) outer parity decoder, incorporated APP channel estimation in dashed rectangle.

The APP phase estimation proceeds as follows:

1. Iteration 1: Send channel metrics $p(y_k|x_k) = (\pi N_0)^{-1}\exp(-|y_k - x_k|^2)$ to inner APP decoder; no channel estimate is available, so $\tilde{h}_k = 1$.
2. Generate APP $p(x_k|y_k)$ from inner APP decoder; use as *a priori* $p(x_k)$ to find $\tilde{x}_k$ according to Equation 9.4.
3. Calculate channel estimates $\hat{h}_k$ according to Equation 9.5.
4. Normalize channel estimates to $\hat{h}_{k,\text{norm}}$ as per Equation 9.6.
5. Filter normalized instantaneous channel estimates $\hat{h}_{k,\text{norm}}$ to provide smoothed estimates $\tilde{h}_k$.
6. Calculate improved channel metrics $p(y_k|x_k, \tilde{h}_k)$ according to Equation 9.7.
7. Next iteration: send improved channel metrics from step 6 to inner APP decoder together with *a priori* soft information from outer APP decoder; go to step 2 or stop with code convergence or at final iteration count.

Averaging is done over the instantaneous channel estimates rather than the instantaneous phase estimates. Addition of the phase estimates may result in loss of accuracy due to the fact that the end result must be taken modulo 2π, and thus the relative importance of the lower order decimal places rises. However, when adding a large number N of terms, the lower order places may be truncated. The channel estimates provide a more stable platform for averaging.

APP channel estimation has two basic components per iteration: 1) an expectation step which calculates $E[x_k]$ and $E[h_k]$; and 2) a maximization step which uses h_k obtained in the expectation step and then filtered to calculate new channel metrics. Thus APP channel estimation can be viewed as basically an expectation-maximization (EM) algorithm.

As mentioned previously, the differential trellis is rotationally invariant to integer multiples of $\pi/4$ rads phase shifts. If, however, in decoding the trellis, the beginning and final trellis states are assumed fixed to state 0 as is commonly done, endpoint errors will occur with a phase shift. The rest of the trellis will shift to a rotated sequence, but the endpoints cannot shift and remain fixed, causing errors. These endpoint errors cause a high flat error floor at a BER of approximately 10^{-4}. To prevent these endpoint errors, a "floating" decoding trellis is used for the inner APP decoder, where both beginning and final states are assumed unknown and set to uniform probabilities.

Figure 9.3 illustrates the rotational invariance of the differential inner code, with a sample random walk phase process at top and the final APP phase estimate beneath. Twice, the phase estimate 'slips' to a phase rotated $-\pi/4$ rads from the random walk phase. However, no decoding errors occur due to these phase slips. The rotationally invariant inner trellis ensures no decoding errors if the phase estimator converges to a rotated phase, providing a parallel decoding path, and the outer parity decoder cancels the bit errors due to phase jumps, so the phase estimator seamlessly slips between constellation symmetry angles.

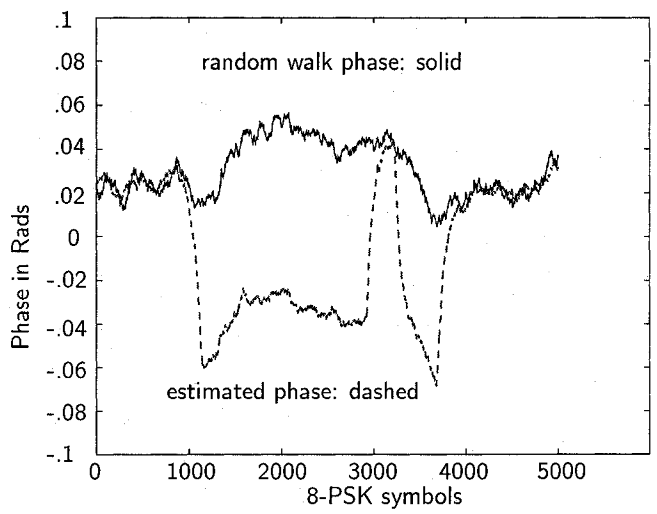


Figure 9.3: Random walk channel phase model and estimated phase with $\pi/4$ phase slips, decoded without errors.

9.5 Channel Phase Models

We consider a channel with possibly time-varying phase offset and unity gain so that $h_k = \exp(j\varphi_k)$. Three different channel phase models are considered:

- Constant Phase Offset: $\varphi_k = \Phi$ for $\Phi = \pi/16$, $\pi/10$ and $\pi/8$ rads.
- Random Walk Phase Process: $\varphi_k = \varphi_{k-1} + \theta_k + \Phi$,
  $\theta_k : \mathcal{N}(0, \sigma_\theta^2)$ for $\sigma_\theta^2 = 0.05$ deg^2.
- Linear Phase Process: $\varphi_k = \varphi_{k-1} + 2\pi\Delta f T$.

All channel models include AWGN.

9.5.1 Constant Phase Offset

With a constant phase offset, $h = \exp(j\Phi) = h_k$, $\forall k = 1, \ldots, N$. For the constant phase offset case, a simple averaging filter is used, with equal coefficients $f_k = 1/N$, as

$$\tilde{h} = \frac{1}{N}\sum_{k=1}^{N} \hat{h}_k, \tag{9.8}$$

where $\hat{h}_k$ is determined from Equation 9.5.

Simulation results of the estimated channel phase for a constant channel phase offset vs. iterations are shown in Figure 9.4 for a phase offset of $\Phi = \pi/16$ rads=11.25^o, $\pi/10$ rads=18^o, and $\pi/8$ rads=22.5^o, all at SNR 3.8 dB, which is a waterfall SNR. A phase offset of $\pi/16$ rads can be compensated for in about 15 iterations at SNR 3.8 dB using APP channel estimation with an averaging filter; a phase offset of $\pi/10$ rads requires 25 iterations at the same SNR, and a phase offset of $\pi/8$ rads has not yet converged by 50 iterations.

A $\pi/8$ channel phase offset may cause the channel phase estimate to completely fail to correctly estimate the offset within a reasonable (< 150) number of iterations, especially at high SNR. This failure occurs when approximately half the symbols decode

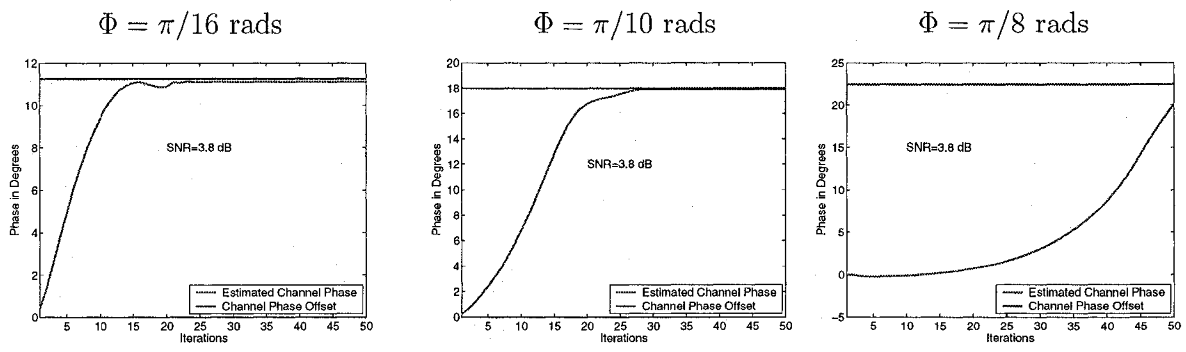


Figure 9.4: Phase estimate vs iterations for constant phase offset=$\pi/16$, $\pi/10$ and $\pi/8$ rads, over AWGN channel at SNR 3.8 dB, using APP channel estimation, with averaging filter; outer (3,2,2) parity/inner D8-PSK SCC with improved mapping and $S = 3$ 15k bit interleaver.

to a phase near around the $\pi/4$ rotated phase offset of $-\pi/8$ radians, and half decode to a phase near the actual phase offset $\pi/8$. These average out to nearly zero phase. The phase acquisition process in this case is lengthened an arbitrarily long time due to the metastability of the $\pi/8$ phase offset. It may take 300 iterations or more for the phase estimate to converge sufficiently to allow accurate symbol decoding. High SNR actually aggravates this situation because the phase estimates are better, and thus more tightly clustered around $\pi/8$ and $-\pi/8$. This effect causes an error floor for $\pi/8$ phase offset, as will be shown in the simulations of Section 9.6.

Use of a smoothing filter with exponentially-decaying filter taps, as in Equation 9.10 for the time-varying phase processes discussed next, instead of the simple averaging filter used for the constant phase offset case, is also examined. The exponential decay of the filter taps could be expected to mitigate the effect of some symbols slipping to the opposite phase.

9.5.2 Random Walk Phase Process

The random walk phase process is a first-order Markov process, where the phase at sampling time kT depends on the phase of the previous time $(k-1)T$ plus a zero-mean

Gaussian-distributed random variable. This random walk phase process is described as

$$\varphi_k = \varphi_{k-1} + \theta_k + \Phi, \tag{9.9}$$

where φ_k is the channel phase offset at sampling time kT, θ_k is a zero-mean Gaussian-distributed phase of variance σ_θ^2, and Φ is a possible constant phase offset. A sample random walk phase process is shown in Figure 9.3 as the solid black curve.

Instantaneous channel estimates $\hat{h}_k$ are found as per Equation 9.5. These estimates are then normalized and filtered through a moving average filter with exponential decay parameter α to obtain

$$\tilde{h}_k = (1-\alpha)\sum_{j=1}^{k} \alpha^{k-j}\hat{h}_j. \tag{9.10}$$

Choice of the exponential decay parameter α determines the effective filter length (filter taps decay faster with smaller α), bandwidth and smoothing (greater smoothing as α approaches 1). Typically, α is close to 1, although $1 > \alpha > 0$. Properties of the exponentially-decaying moving average (MA) filter are explored further in Appendix B.

The rotational invariance of the differential 8-PSK trellis to multiples of $\pi/4$ rads phase rotation is vividly displayed in Figure 9.3. The random walk channel phase is displayed in solid black at top, while the APP filtered channel phase estimate is the dashed curve beneath it. The phase estimate clearly slips twice to a phase rotated by $-\pi/4$ rads from the actual channel phase. Interestingly, there are no errors, even at the phase discontinuities. This is due in part to the rotationally invariant differential 8-PSK trellis, which does not generate errors along the rotated phase sequence, but only at the phase discontinuities or phase slips. These few bit errors are corrected by the outer code.

9.5.3 Linear Phase Offset

A constant frequency/linear phase offset could model oscillator drift or a mobile Doppler scenario. The carrier frequency f_c replicated at the receiver to downconvert the received passband signal to baseband frequencies is generated by a voltage-controlled oscillator

(VCO). As the VCO ages, its frequency drifts; high-quality oscillators may have a drift of 0.1 part per million (ppm), while mediocre oscillators can drift more than 1 ppm. There may also be an uncorrected frequency offset between transmitter and receiver.

In a mobile Doppler scenario, a frequency offset at the receiver occurs when the transmitter is moving relative to the receiver or vice versa. This frequency offset, termed the Doppler shift, is found as $\Delta f = v f_c / c$, where v is the transmitter velocity, and c is the speed of light.

The linear phase process assumes a phase change of $\Delta f T$ per symbol, where T is the symbol transmission time. The same exponentially-decaying moving average filter described for the random walk time-varying process is used for the linear phase offset as well.

9.6 Simulations of APP Channel Phase Estimation

Simulation results are presented for the three phase models described above, using APP phase estimation integrated into the iterative decoding process. All channel phase estimation results use the same S-random interleaver with minimal spreading $S = 3$, size 15000 bits, and are compared with coherent results for the same interleaver.

Note that coherent decoding results are slightly better than for the random interleaver results of Chapter 7, which generate a new random interleaver design with each codeword frame. Random interleavers are good for examining the average performance of a concatenated coding system. However, they typically provide poorer performance than a interleaver designed for specific distance properties, as some random interleavers will contain low-weight error events, with poorer performance. In practice, of course, a specific interleaver design must be used at both transmitter and receiver, and stored either in memory or generated anew, if based on an algebraic design, at the receiver. For simplicity and increased simulation speed, the same $S = 3$ 15 kbit S-random interleaver is used here for all codeword frames of all channel estimation simulations.

9.6.1 Constant Phase Offset

Figure 9.5 compares the BER performance of decoding with and without CSI over an AWGN channel with constant channel phase offset of $\pi/16$, $\pi/10$ and $\pi/8$ rads. Fifty decoding iterations are used. The constant phase channel estimate $\tilde{h}$ for the entire frame is an average of the instantaneous channel estimates $\hat{h}$ as described in Section 9.4. This averaging filter is denoted as 'fltr A' in Figures 9.5 and 9.7. Near-coherent-decoding results are obtained for phase offsets of $\pi/16$ and $\pi/10$ rads.

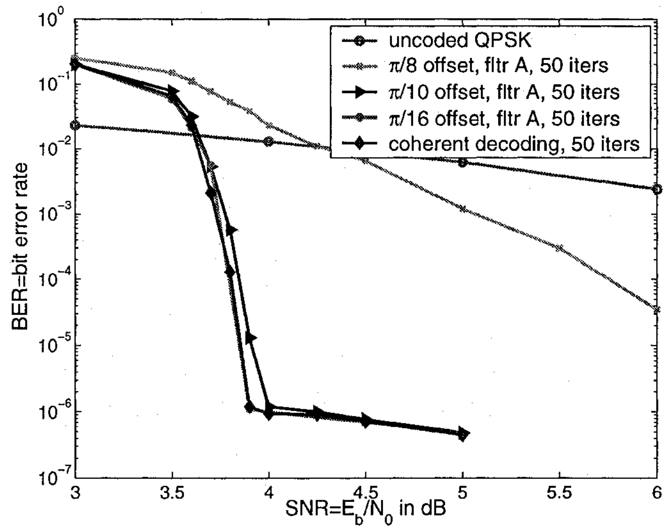


Figure 9.5: BER results without CSI, phase offset=$\pi/16$, $\pi/10$ and $\pi/8$ rads, AWGN channel, using APP channel estimation, compared to coherent decoding; S=3 15k bit interleaver, improved 8-PSK mapping.

Performance degrades as the phase offset approaches $\pi/8$. This is a metastable point, as $\pi/8$ is halfway between two valid differential sequences. The instantaneous channel estimates $\hat{h}$ oscillate to either side of the $\pi/8$ boundary, and convergence to the correct phase estimate is very slow for a phase offset of exactly $\pi/8$ rads. As shown in Figure 9.5, a significantly raised error floor exists with a phase offset of exactly $\pi/8$ rads for 50 decoding iterations, along with 2+ dB loss in waterfall SNR. Even at higher SNR, decoding has still not converged for erroneous frames by 50 iterations. This can be seen by plotting BER vs iterations for a phase offset of $\pi/8$ rads=22.5^o at an SNR of 5.5 dB, as in Figure 9.6. Clearly, the BER is still dropping by 50 iterations; thus the floor is caused by error events that have not yet been corrected within the 50 decoding iterations. The phase estimate in degrees vs iterations is also shown in Figure 9.6. The phase estimate in this example has still not converged to the correct value by 50 iterations, though it is approaching the correct value.

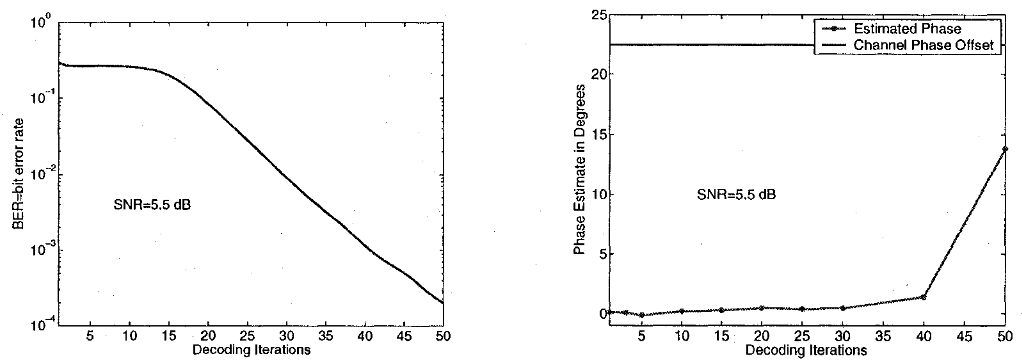


Figure 9.6: BER and phase estimate vs decoding iterations at SNR 5.5 dB with phase offset=$\pi/8$ rads, AWGN channel, using APP channel estimation; S=3 15k bit interleaver, improved 8-PSK mapping.

Increasing the number of decoding iterations from 50 to 100 does improve the performance with a $\pi/8$ offset, as shown in Figure 9.7, by about 1.25 dB. A significantly raised error floor still exists, however. Use of a time-varying filter with exponentially decaying filter taps is also compared in Figure 9.7 for 50 decoding iterations; this is denoted as 'fltr B' in Figure 9.7. Performance is improved, especially in the error floor region, with the time-varying filter as compared to the averaging filter of 'fltr A'. However, the $\pi/8$ phase offset with 'fltr B' still has 0.75-0.9 dB loss in waterfall performance and a raised error floor compared to coherent decoding, due to lack of convergence within 50 iterations even at higher SNR.

9.6.2 Random Walk Phase Offset

APP channel phase estimation results for a random walk phase process with $\sigma_{\Phi}^2 = 0.05\ \text{deg}^2$ are compared to coherent decoding results in Figure 9.8. Fifty iterations

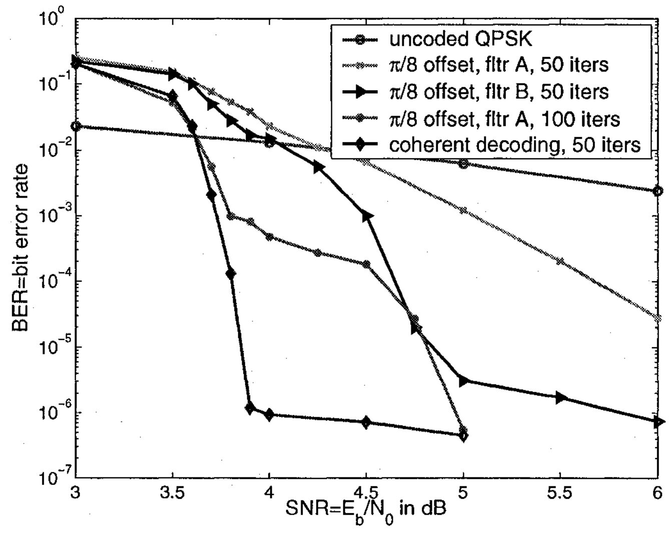


Figure 9.7: BER results without CSI, phase offset=$\pi/8$ rads, AWGN channel, using APP channel estimation, comparing averaging filter and moving average filter, and increasing iterations; S=3 15k bit interleaver, improved 8-PSK mapping.

are used. The channel phase estimation filter coefficients are given by $\tilde{h}_k = (1 - \alpha)\sum_{j=1}^{k} \alpha^{k-j}\hat{h}_j$ with exponential decay parameter $\alpha = 0.99$.

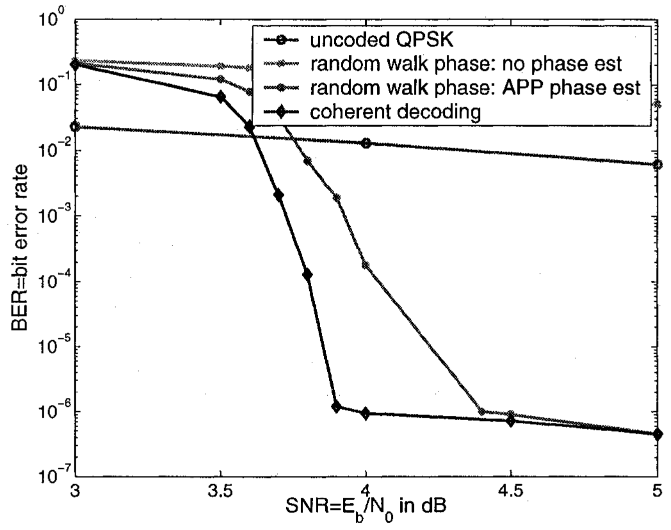


Figure 9.8: BER results without CSI, random walk phase processes using APP channel estimation compared to coherent decoding; interleaver size=15k bits, S=3 interleaver, improved 8-PSK mapping.

APP channel estimation for the random walk phase process gives results 0.25-0.5 dB from coherent decoding along the turbo cliff, where 50 iterations are insufficient for convergence. At higher SNR, coherent decoding performance is achieved using APP channel estimation with the exponentially-decaying moving average filtering of the instantaneous channel estimates. For comparison, BER curves for both uncoded QPSK and iterative decoding of the (3,2,2)/D8-PSK SCC without any phase estimation are shown as well. Without phase estimation, iterative decoding fails, and performs worse than uncoded QPSK for the random walk phase offset.

9.6.3 Linear Phase Offset

A constant frequency/linear phase offset may model oscillator drift or a mobile Doppler scenario. A carrier frequency f_c of 1 GHz and oscillator drift of 0.1 ppm from a high quality oscillator gives Δf=100 Hz. A symbol rate of 10^6 symbols/sec corresponds to $\Delta fT = 10^{-4}$. A lower carrier frequency of 450 MHz reduces ΔfT to 4.5×10^{-5}.

A typical highway velocity of 110 km/h and a carrier frequency of 1 GHz correspond to a Doppler shift Δf of 100 Hz. Increasing f_c to 2.1 GHz, a 3G/CDMA/GSM allocated downlink frequency, gives a Doppler shift of 210 Hz, and $\Delta fT = 2.1 \times 10^{-4}$.

Three values of constant frequency/linear phase offset are considered: $\Delta fT = 4.5 \times 10^{-5}$, 10^{-4}, and 2.1×10^{-4}, with larger ΔfT expected to negatively impact performance.

Simulation results for a linear phase process with $\Delta fT = 4.5 \times 10^{-5}$, 10^{-4}, and 2.1×10^{-4} are also shown in Figure 9.9.

APP phase estimation results are approximately 0.5 dB from coherent decoding for $\Delta fT = 2.1 \times 10^{-4}$, with 0.4 dB loss for $\Delta fT = 10^{-4}$, and near coherent performance for $\Delta fT = 4.5 \times 10^{-5}$. The largest frequency offset has a slightly raised error floor, but smaller offsets converge to the coherent decoding error floor. BER performance curves for uncoded QPSK and for the smallest frequency offset, $\Delta fT = 4.5 \times 10^{-5}$, iteratively decoded without APP phase estimation, are shown for comparison. Iterative decoding fails totally at even the smallest frequency offset case when no phase estimation

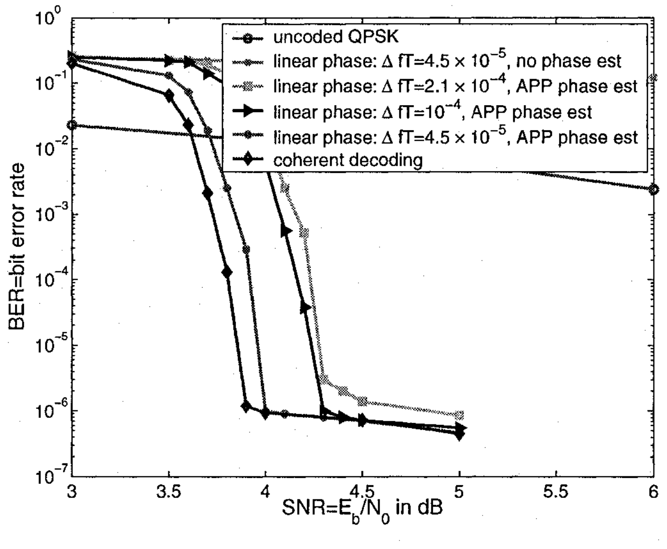


Figure 9.9: BER results without CSI, linear phase processes using APP channel estimation compared to coherent decoding; interleaver size=15k bits, S=3 interleaver, improved 8-PSK mapping.

is performed, due to the frequent phase slips.

A total of 100 decoding iterations are performed on each codeword frame. An increased number of decoding iterations are required for good performance at larger ΔfT because the phase estimate will slip at every $\pi/4$ phase increase. Each slip is a MSED symbol error, with corresponding bit errors. The outer parity code can correct bit errors that map to isolated (3,2,2) codewords, but with several phase slips, bit errors may map two to a codeword, which cannot be corrected. This causes a significantly raised error floor.

A sample phase estimate for $\Delta fT = 5 \times 10^{-4}$ is shown for 50 decoding iterations in Figure 9.10, along with associated bit errors shown in their corresponding symbol positions. The linear channel phase offset is shown in the straight line, changing by π radians every 1000 symbols. The final channel phase estimate after 50 decoding iterations is shown below, with multiple phase slips The bit errors are the short vertical lines, and their location indicates a bit error in the corresponding 8-PSK input symbol. The channel phase estimate clearly does not directly track the channel phase offset, but slips several times in every π radians/1000 symbol interval. These slips occur at $\pi/4$ or $\pi/8$ intervals, or multiples thereof, at the boundaries between symbols.

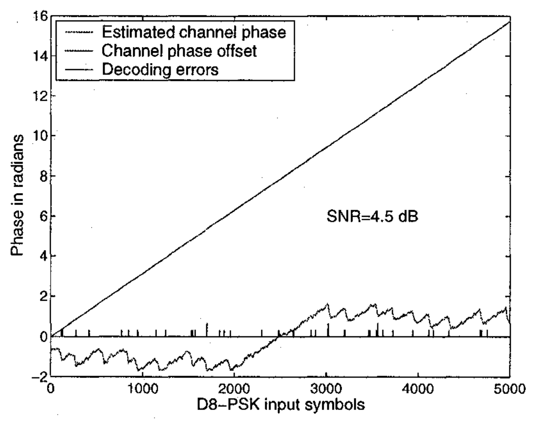


Figure 9.10: Linear phase offset, $\Delta fT = 5 \times 10^{-4}$, and channel phase estimate after 50 iterations at SNR 4.5 dB with APP channel estimation; outer (3,2,2)/inner D8-PSK SCC with improved mapping, $S = 3$ 15 kbit S-random interleaver.

This slippage can be viewed easily when considering the all-zeros sequence, where the actual symbol phase is always 0^o. When transmitted through a channel with a linear phase offset and no noise, the received sequence will have the linear phase of the offset. The D8-PSK decoder would then decode to a sequence with increasing symbol phase every $\pi/4$ radians; however, this slip may occur at the $\pi/8$ boundary between symbols, or further into the new symbol. The decoder cannot tell the difference between an all-zeros sequence $\mathbf{x} = [e^{j0}e^{j0} \ldots e^{j0} \ldots e^{j0}]$ with linear phase offset, and the sequence $\hat{\mathbf{x}} = [e^{j0}e^{j0} \ldots e^{j\pi/4}e^{j\pi/4} \ldots e^{j\pi/2}e^{j\pi/2} \ldots]$ with no phase offset, where the phase slips by $\pi/4$ somewhere at or after the $\pi/8$ boundary between symbols. Errors will occur at each boundary slip. Depending on their de-interleaved positions in the (3,2,2) parity code, these errors may or may not be corrected by the outer decoder. More boundary slips result in an increased number of errors.

An additional enhancement of the channel estimate filter is incorporated to speed up convergence in the linear phase case, although 100 decoding iterations are needed for good performance; without this enhancement, 150 decoding iterations would be required for similar performance. The APP channel estimation of the linear phase process uses two different filter parameters, α_0 for use initially and α_1 for use later when the symbol estimates and channel estimates are for the most part good, with the exception of the phase slip errors. The decision of when to switch from α_0 to α_1 is made based on the value of the mean *a priori* LLR magnitude $\mu_{|\lambda_a(v')|}$ into the D8-PSK APP decoder.

A small $\mu_{|\lambda_a(v')|}$ means that the APP information and channel estimates are still poor, while a large $\mu_{|\lambda_a(v')|}$ indicates the estimates have high reliability. Observation of both the number of bit errors and $\mu_{|\lambda_a(v')|}$ shows that when $\mu_{|\lambda_a(v')|} \geq 10$, most bit errors have been corrected. This value is SNR-dependent, as the initial expected LLRs are SNR-based, and increase as SNR increases. However, we are interested in the error floor performance, or SNR > 4 dB, and thus values appropriate for this region should be appropriate for higher SNR as well. Therefore, the decision threshold for switching from α_0 to α_1 is $\mu_{|\lambda_a(v')|} \geq 10$.

If too much smoothing is applied initially ($\alpha_0 \approx 1$), so that channel estimates from phases across multiple $\pi/8$ regions are combined, the phase estimate will not converge to even a phase-slipped version. This is similar to the problem seen with a constant phase offset of $\pi/8$. The initial smoothing should thus use a smaller value of α_0 which decays quickly. Once the phase estimate is near convergence, as indicated by increased $\mu_{|\lambda_a(v')|}$, then a larger value for $\alpha_1 > \alpha_0$ can be used to speed up convergence.

For $\Delta fT = 4.5 \times 10^{-5}$ and 10^{-4}, the exponential decay parameters used were $\alpha_0 = 0.975$ and $\alpha_1 = 0.99$; for $\Delta fT = 2.1 \times 10^{-4}$, $\alpha_0 = 0.95$ and $\alpha_1 = 0.99$ were used. Larger linear phase offsets, with a greater number of phase slips in the estimate, see better performance from smaller values of α_0, which reduces the effective filter size and smoothing initially.

Chapter 10

APP Timing Estimation

10.1 Introduction

Timing, or symbol, synchronization is a necessary component of receiver function. The transmitter sends symbols, multiplied by a shaped pulse $p(t)$, at a rate $1/T$. The receiver samples the received signal at a rate $1/T$ or multiples thereof. However, the receiver clock is not initially synchronized to the transmitter clock. Due to this timing discrepency between transmitter and receiver, sampling may not occur at the ideal time, which is at the peak of the symbol pulse, causing loss of signal strength and interference from adjoining symbol pulses, known as intersymbol interference (ISI) and discussed in more detail in Section 10.2.

The need for timing synchronization between transmitter and receiver comes about because the receiver should sample not just somewhere within each symbol period, but ideally at the maximum of the pulse. The time-domain symbol pulses used in practice are not the idealized rectangular pulse or square wave shape sometimes assumed in theory. Rectangular pulses are not practical from several standpoints: an approximate rectangular pulse with short rise time is very difficult to build, suffers from ringing at the pulse edges (Gibbs phenomenon), and has near-infinite bandwidth (a true square wave in the time domain has infinite bandwidth). Bandwidth-limited communications systems need to use bandwidth-limited pulses and cannot afford high-bandwidth transmission symbol pulses.

However, band-limited pulses have infinite time extent, and converge only asymptotically towards zero. Each pulse has significant overlap into other symbol transmission periods. Pulse shapes that converge faster will interfere less with surrounding pulses when timing synchronization has not yet been achieved.

10.2 Intersymbol Interference (ISI)

Consider the sequence of pulses shown in Figure 10.1A. Only 5 pulses are considered for easy visibility. These pulses are raised cosine pulses, defined later in Section 10.3, a pulse type commonly used in practice due to their band-limited frequency response. The sampling times are shown as vertical bars. The transmission time between pulses is T. If the receiver samples this sequence at integer multiples of time T, each pulse is sampled at its maximum as shown in the diagram, and none of the other pulses interfere. However, if the receiver clock is offset with respect to the transmitter clock by a small amount ϵ, say $0.4T$ as shown in Figure 10.1B, each pulse is not only now sampled below its maximum, but also the other pulses interfere significantly with the pulse of interest. This interference is known as **intersymbol interference** (ISI). Clearly, timing synchronization is critical to the system performance in reducing ISI as an additional source of noise interference.

Timing synchronization is normally performed as a separate operation before the decoding block. Just as with APP channel estimation, APP timing estimation incorporates the timing estimation within the decoding block, using the inner D8-PSK APP symbol probabilities $P(\mathbf{x}|\mathbf{y})$ available with each iteration to generate an iteratively-improving timing error estimate. The timing error estimate is used to calculate, via interpolation, an approximation of the received sequence sampled at the correct time (if the timing error estimate equals the timing offset). This more accurate received sequence is then used to generate new channel metrics as input for the inner D8-PSK APP decoder in the following iteration.

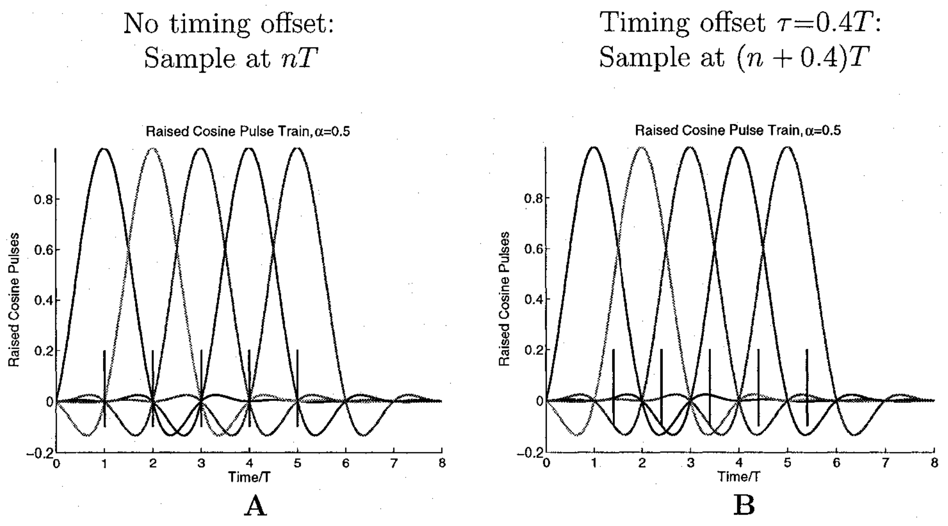


Figure 10.1: A sequence of 5 sinc pulses, sampled at nT and with an offset of $0.4T$.

In order to discuss timing estimation algorithms, a representation for the received signal with timing offset is necessary. The type of baseband pulse utilized for signalling is required, as the pulse shape determines the exact amount of ISI incurred for a given timing offset. The baseband pulse was not needed for the APP phase estimation, as timing synchronization was assumed already achieved prior to decoding.

10.3 Nyquist Band-limited Pulses

The Nyquist bandwidth constraint, or Nyquist ISI constraint, is a requirement which the signal pulse frequency response $P(\omega)$ must meet to avoid ISI at the receiver when timing synchronization is perfect. The Nyquist bandwidth constraint is:

$$\sum_{l=-\infty}^{\infty} P(f - l/T) = T, \tag{10.1}$$

where T is the symbol transmission time. Pulses which satisfy Equation 10.1 are known as Nyquist pulses. Note that the rectangular time-domain pulse, with a sinc frequency response which is zero at non-zero integer multiples of $1/T$, satisfies the Nyquist bandwidth constraint although its frequency response is not band-limited.

Clearly a rectangular pulse frequency response which is band-limited to $\pm 1/2T$ satisfies Equation 10.1, and is in fact the minimum bandwidth pulse which satisfies the Nyquist ISI criterion. The rectangular frequency response has a sinc time-domain pulse, which is zero at non-zero integer multiples of T, and thus there is no ISI as long as timing synchronization is achieved. However, the sinc pulse is non-causal and has infinite time duration, dying out as $1/t$ [74]. When a timing offset exists between transmitter and receiver, ISI is significant with the sinc pulse. Therefore the sinc pulse is not practical for actual communications systems.

A family of Nyquist band-limited pulses commonly used are the raised cosine, or cosine rolloff, pulses. A raised cosine pulse $p_{RC}(t)$ in the time domain is described by

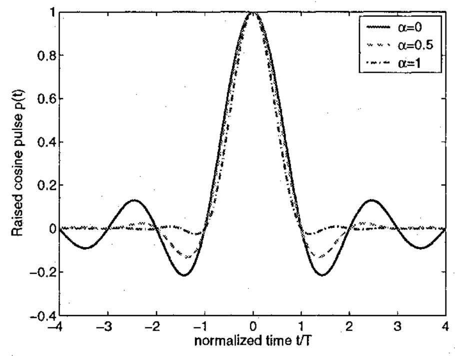
$$p_{RC}(t) = \frac{\sin(\pi t/T)}{\pi t/T} \frac{\cos(\alpha \pi t/T)}{(1 - 4\alpha^2 t^2/T^2)}, \tag{10.2}$$

where α is the rolloff factor, ranging from $0 \leq \alpha \leq 1$. For $\alpha = 0$, $p_{RC}(t)$ is a sinc function, and for $\alpha = 1$, $p_{RC}(t)$ is a square wave. The frequency response $P_{RC}(w)$ for a raised cosine pulse is given by

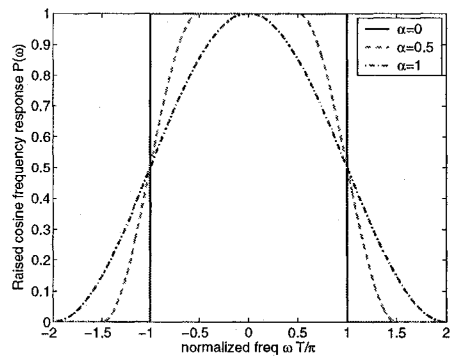
$$P_{RC}(\omega) = \begin{cases} T, & 0 \leq |\omega| < \pi(1-\alpha)/T; \\ T/2 \left[1 - \sin\left(\frac{|wT| - \pi}{2\alpha}\right)\right], & \pi(1-\alpha)/T \leq |\omega| < \pi(1+\alpha)/T; \\ 0, & |\omega| \geq \pi(1+\alpha)/T. \end{cases} \tag{10.3}$$

Raised cosine pulses are a good compromise between minimum bandwidth and rapid pulse die-down outside its symbol period, and are often used in practical communications systems. The raised cosine pulse bandwidth $W_{RC} = (1+\alpha)/(2T)$ is increased over the minimum bandwidth $1/(2T)$ by $\alpha/(2T)$.

Figure 10.2A shows the pulse $p_{RC}(t)$ for both a sinc function (raised cosine with 0% rolloff), and raised cosine pulses with 50% and 100% rolloff; the x-axis is normalized time t/T. Figure 10.2B displays the frequency response $P_{RC}(w)$ for these same raised cosine pulses, with the x-axis being normalized frequency $\omega T/\pi$. Notice that $p_{RC}(t) = 0$ for all $t = lT, l = 1, 2, \ldots$ for each pulse. Also note the rapid die-off in time of $p_{RC}(t)$ as the rolloff factor α increases. In general, for $\alpha > 0$, the tails of $p_{RC}(t)$ die off as $1/t^3$ [74]. A timing error in sampling still leads to ISI components, but the interfering pulses have reduced magnitude compared to the sinc pulses.



A



B

Figure 10.2: Raised cosine pulse response in the time domain and frequency domain, for varying rolloff α.

There are other baseband pulses used in practice as well, but raised cosine pulses are the most common. Often the raised cosine pulses are split between transmitter and receiver, so that the root raised cosine (RRC) pulse is used for transmission. The root raised cosine (RRC) pulse frequency response $P_{RRC}(\omega)$ is simply the square-root of the raised cosine pulse $\sqrt{P_{RC}(\omega)}$. The receiver RRC filter serves as a pulse-matched filter, and the combination of the transmit RRC pulse and RRC matched filter together make a raised cosine filter.

To simulate performance of a timing synchronization algorithm, a particular baseband pulse must be chosen. Neither the time-domain sinc pulse ($\alpha = 0$), with brickwall frequency response, or the brickwall time-domain pulse ($\alpha = 1$) are practical to implement. This thesis uses a rolloff factor of $\alpha = 0.5$, or 50% rolloff, as a reasonable compromise and a commonly-implemented pulse shape. With 50% rolloff, the pulse bandwidth increases in frequency to $3/(4T)$ instead of $1/(2T)$ for a brickwall time-domain pulse.

10.4 Nyquist Sampling Rate

The Nyquist-Shannon sampling theorem [108], [109] states the requirements for a signal $p(t)$ to be fully reconstructed from a sampled version $p(nT_s)$ of that signal, sampled every T_s seconds. If the maximum frequency of the continuous-time signal $P(\omega)$ is W, then the sampling rate $1/T_s$ must be greater than $2W$ for complete reconstruction of the continuous signal from the sampled signal. The Nyquist sampling rate is given as

$$\begin{aligned} \frac{1}{T_s} &> 2W, \\ T_s &< \frac{1}{2W}. \end{aligned} \tag{10.4}$$

For a raised cosine pulse with 50% rolloff, the sampling rate $1/T_s$ and sampling time T_s must be

$$\begin{aligned} \frac{1}{T_s} &> 2(3/(4T)) = 1.5/T \\ T_s &< \frac{1}{1.5/T} = \frac{T}{1.5} \end{aligned} \tag{10.5}$$

In practice, this fractional sampling can be difficult to implement, resulting in some symbols being sampled twice and some once if uniformly-spaced sampling is used. Sampling once per symbol is not sufficient for Nyquist sampling of the raised cosine pulse with $\alpha > 0$, but sampling twice per symbol is sufficient for all $1 \geq \alpha > 0$. This thesis implements a sampling rate of twice per symbol, $T_s = T/2$.

Several timing estimation algorithms utilize two samples per symbol, so this requirement for increased sampling rate due to the increased bandwidth of the 50% rolloff raised cosine pulse is not a major handicap.

10.5 Transmitted Signal

The baseband signal $s(t)$ of the symbols $x_0, \ldots, x_N$ (in this system, D8-PSK symbols) multiplied with the pulse-shaping filter $p(t)$ is given by

$$s(t) = \sum_l x_l p_{RRC}(t - lT) \tag{10.6}$$

This baseband signal is up-converted to the carrier frequency ω_c by multiplication with the carrier signal $\exp(j\omega_c t)$.

10.6 Received Signal

The received signal $r(t)$ is down-converted to baseband by multiplying with a carrier signal generated by its local oscillator (LO) replica of the carrier $\exp(j(\omega_c t)$. Any offset in carrier frequency results in a linear phase shift, as explored in the previous Chapter 9. In this chapter, the receiver-generated replica of the carrier ω_c is assumed to be exact. There could also be an existing channel phase offset. This is assumed to be compensated for by phase estimation prior to the decoding process. Only timing estimation is considered in this chapter.

Low-pass filtering is required to remove the higher harmonics resulting from multiplying the carrier-modulated signal with a replica of the carrier. When pulse-matched filtering is done at the receiver, because the symbol pulses are specifically designed to be band-limited, the matched filter is a low-pass filter. Thus matched filtering is sufficient to remove the higher frequencies from the signal, resulting in a baseband signal at the output.

The output of the matched filter is sampled every T_s seconds. For a symbol transmission time T and raised cosine pulse with rolloff > 0, the sample time is $T_s = T/2$, for a sampling rate of twice per symbol. If transmitter and receiver clocks are synchronized, the timing offset $\tau = 0$, and the receiver samples the output of the RRC-matched filter at integer values of T_s relative to the transmit pulse. Since the frequency response of the squared RRC is simply the RC pulse, the sampled output of the filter is the sequence of RC pulses sampled at each time kT_s. The sampled, matched-filtered output $y(kT_s)$ is given by

$$y(kT_s) = \sum_l x_l p_{RC}(kT_s - lT) + n(kTs), \tag{10.7}$$

where n is zero-mean AWGN with variance σ_n^2. For k even, the samples are at integer values of T, and there are no interfering pulses, so $y(kT_s) = x(kT/2) + n(kT/2),\ k$ even. For k odd, the samples are shifted by $T/2$ from the integer values of T, and the pulses interfere.

If transmitter and receiver clocks are not synchronized, there will be a timing offset τ between the transmit pulse and the receiver samples. The sampled, matched-filtered output $y(kT_s + \tau)$ is

$$y(kT_s + \tau) = \sum_l x_l p_{RC}(kT_s + \tau - lT) + n(kTs + \tau). \tag{10.8}$$

As seen in Figure 10.1, when a raised cosine pulse is sampled at times $kT + \tau$ offset from the symbol timing, adjoining pulses add in to the pulse transmitted at time kT. These adjoining pulses are all pulses of Equation 10.8 where $l \neq 0$.

10.7 Timing Offset Model

The timing offset τ between transmitter and receiver is considered here to be constant over the length of a codeword frame (5000 symbols for the (3,2,2) parity/D8-PSK SCC). At first, a constant timing offset over all frames is considered, similar to the constant phase offset. As a basic timing offset model, it illustrates whether the APP timing estimation technique functions at all. Next, a timing offset which is constant over one frame but with uniformly random distribution over all codeword frames is examined.

10.8 Maximum Likelihood Timing Estimation

As shown above, timing sychronization between transmitter and receiver is necessary for best reception. Timing estimation algorithms seek to achieve timing synchronization by estimating the timing offset τ between transmitter and receiver, often according to the maximum likelihood (ML) criterion.

Maximum likelihood, or ML, decoding consists of choosing the estimated symbol sequence $\hat{\mathbf{x}}$ such that the *a posteriori* probability is maximized as in Equations 2.3-2.5 to obtain

$$\hat{\mathbf{x}} = \arg\max_{\mathbf{x}} p(\mathbf{y}|\mathbf{x}) \tag{10.9}$$

ML decoding is optimal decoding in the sense that it minimizes the sequence error probability.

A timing offset τ is a synchronization parameter included as $p(\mathbf{y}|\mathbf{x}, \tau)$. Typically, the phase θ is also a synchronization parameter; however, this chapter assumes that the phase parameter θ has already been estimated. Joint timing and phase synchronization is considered in Chapter 11. The effect of the timing offset τ can be removed by integrating over τ if the distribution of τ is known (typically assumed uniform unless known differently):

$$p(\mathbf{y}|\mathbf{x}) = \int_{\tau} p(\mathbf{y}|\mathbf{x}, \tau)p(\tau)d\tau. \tag{10.10}$$

At high SNR, the likelihood function $p(\mathbf{y}|\mathbf{x}, \tau)$ weighted by the *a priori* $p(\tau)$ is assumed to concentrate at its maximum values, *i.e.* maximizing the integral is well-approximated by maximizing the integrand. If $p(\tau)$ is assumed to be uniform, then it is constant over all τ, has no effect on the maximization and can be removed, as

$$\hat{\mathbf{x}}, \hat{\tau} = \arg\max_{\mathbf{x},\tau} p(\mathbf{y}|\mathbf{x}, \tau) \tag{10.11}$$

10.9 Timing Estimation Algorithms

10.9.1 Decision-directed and Non-data-aided Algorithms

There are three main categories of timing synchronization or estimation algorithms:

1. **Class DD - Decision-directed:**

   The estimated sequence $\hat{\mathbf{x}}$ is used as the correct sequence $\mathbf{x}$, assuming that $P(\mathbf{x} = \hat{\mathbf{x}}) \simeq 1)$ and thus

   $$\hat{\tau}_{DD} = \arg\max_{\tau} p(\mathbf{y}|\mathbf{x} = \hat{\mathbf{x}}, \tau) \tag{10.12}$$

   An initial estimate of $\mathbf{x}$ is required for DD algorithms. This initial estimate can be provided either by a hard decision on input samples or soft decisions from a decoder, as proposed in this thesis.

2. **Class DA - Data-aided:**

   A known training sequence $\mathbf{x}_0$ is sent during acquisition and τ is determined by

   $$\hat{\tau}_{DA} = \arg\max_{\tau} p(\mathbf{y}|\mathbf{x} = \mathbf{x}_0, \tau) \tag{10.13}$$

3. **Class NDA - Non-data-aided**

   Nothing is known about the transmitted sequence $\mathbf{x}$, so all sequences are assumed equally likely and averaged over.

   $$\hat{\tau}_{NDA} = \arg\max_{\tau} \sum_{\mathbf{x}} p(\mathbf{y}|\mathbf{x}, \tau) \tag{10.14}$$

As the APP decoder provides extrinsic soft information about the symbols $\mathbf{x}$, choosing a DD timing estimation algorithm which can incorporate this information in the form of estimated symbols $\hat{\mathbf{x}}$ allows the timing algorithm to improve its estimate with each iteration of the decoding loop.

10.9.2 Feedforward and Feedback Algorithms

Estimation algorithms may also be classified as one of two types: feedforward algorithms and feedback, or error-tracking, algorithms. Feedforward algorithms directly estimate the timing offset $\hat{\tau}$ from the received signal before it is corrected with interpolation. Most commonly this is done through a search for the value of τ that maximizes the probabilities in Equations 10.14-10.16, or a related likelihood function derived from them.

The turbo code structure, where the entire block of sampled data and soft symbol information are available during each iteration for estimation of the timing offset, and correction of the sampled data via interpolation for the estimated offset does not occur until after estimation, lends itself to a feedforward structure. However, the feedforward algorithms primarily do a maximal search, whose complexity depends on the quantization of τ for the search, but requires approximately D times as many operations as a feedback algorithm, if τ is discretized into D values. This search would be performed every iteration.

Additionally, to determine the maximum value of the likelihood function requires interpolating the samples at each discrete value of τ. Interpolation is time-consuming, as it consists of filtering the samples through either an ideal interpolator (sinc function in time domain) or one of the simpler but less ideal interpolators discussed in Section 10.12.

For example, a simple NDA feedforward ML timing estimation algorithm chooses the estimated timing offset $\hat{\tau}$ as

$$\hat{\tau} = \arg\max_{\tau_D} \sum_k |y(kT+\tau-\tau_D)|^2 = \arg\max_{\tau} \sum_k y(kT+\tau-\tau_D)y^*(kT+\tau-\tau_D) \quad (10.15)$$

where the original symbol samples are $y(kT + \tau)$, τ_D are the D discretized values of timing offset, and the interpolated samples are at $y(kT + \tau - \tau_D)$. This algorithm can operate at one sample per symbol. Assume there are k symbols, D discretized values of τ, and I taps on the interpolator. To calculate each new interpolated sample at one

τ_D value requires I multiplications and additions, and calculating all new interpolated samples for all values of τ_D requires kID multiplications and additions. Granted, one set of interpolated values is required as output for the best choice of τ_D for any timing estimation algorithm. However, excluding that set of kI multiplies and adds still leaves an additional $kI(D-1)$ multiplies and adds. In comparison with feedback algorithms later, both will include the kI multiplies and adds required for final interpolated values.

The timing estimation algorithm itself requires k multiplications, additions and conjugations per τ_D value, or kD multiplications, additions and conjugations for all τ_D, plus D compares. In total, this FF NDA timing estimation algorithm uses $k(I+1)D$ multiplies, $k(I+1)D$ adds, kD conjugations and D compares per iteration. Comparison with a feedback algorithm later in Section 10.10 will show a factor of $D/2$ difference in computational complexity. Only for $D = 2$, a very coarse discretization for τ_D, are the complexities equivalent between feedforward and feedback timing estimation algorithms.

This thesis does not consider feedforward algorithms further; their higher complexity and latency are counter to the low complexity component code choices.

Feedback algorithms operate with a feedback loop; the algorithm derives an error signal ϵ, such that

$$\epsilon \propto \hat{\tau} - \tau, \tag{10.16}$$

where τ is the actual timing offset and $\hat{\tau}$ is the estimated timing offset. This error signal is typically obtained by differentiating a likelihood function $L(\mathbf{x}, \tau)$ based on the ML probabilities; the value of the derivative $\partial L/\partial \tau$ at the most recent estimate $\hat{\tau}_k$ gives the sign of the current change in timing estimate. The new timing estimate $\hat{\tau}_{k+1}$ is given by

$$\begin{aligned} \hat{\tau}_{k+1} &= \hat{\tau}_k + \beta \frac{\partial L(\hat{\mathbf{x}}_k, \hat{\tau}_k)}{\partial \tau} \\ &= \hat{\tau}_k + \beta \epsilon_k \end{aligned} \tag{10.17}$$

where β is a constant that determines the tracking response. If the likelihood function L is maximized for $\hat{\tau}_k$, so the estimated timing offset $\hat{\tau}$ equals the actual timing offset τ, then the derivative $\frac{\partial L}{\partial \tau}|_{\hat{\tau}_k} = 0$, the timing error signal $\epsilon_k = 0$, $\hat{\tau}_{k+1} = \hat{\tau}_k$, and the timing estimation algorithm locks onto the correct timing offset.

The delay inherent in feedback algorithms is not a problem for the turbo decoder, which operates on an entire block or codeword frame of symbols. The D-8PSK APP decoder supplies APP probabilities for each symbol in the block at the completion of its decoding process for each iteration.

A block diagram of a feedback timing error detector is shown in Figure 10.3. The interpolator calculates updated samples $y(kT + \tau - \hat{\tau}_{k+1})$ based on the current timing estimate $\hat{\tau}_{k+1}$. If timing recovery is performed before decoding, then the TED receives a new sample, fine-tunes its timing estimate based on that sample, and sends the new estimate to the sampling clock which adjusts the sampling for the next sample appropriately. The interpolator corrects the current sample based on the current timing estimate and sends that updated sample to the decoder.

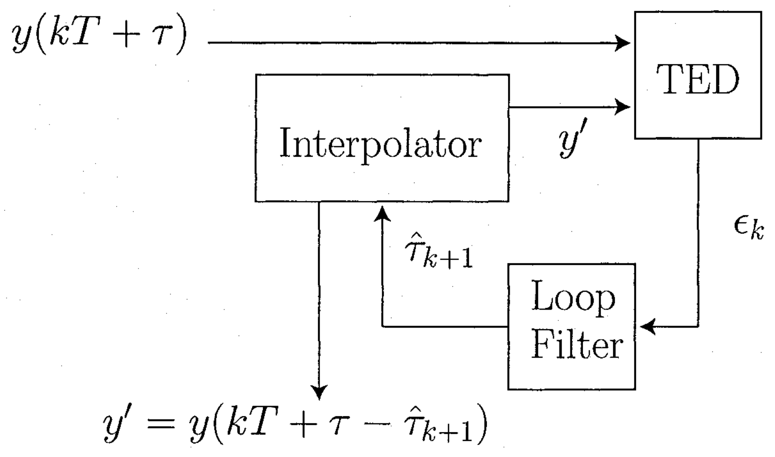


Figure 10.3: Block diagram of a feedback timing error detector loop.

If timing recovery is incorporated into the iterative decoding loop, then the TED works with a block of already-sampled data, rather than a stream of continuous inputs. In the iterative decoding case, the TED will generate a new timing estimate based on the entire block of data, and the interpolator will adjust the sampling time for the entire block. This procedure repeats itself each iteration.

As the timing offset model used in this thesis is τ constant over a codeword frame, instead of calculating $\hat{\tau}_k$ in Equation 10.17, a single estimated offset $\hat{\tau}^i$ is calculated for the entire frame with every iteration i. A single timing error estimate ϵ^i is obtained for each iteration by averaging the individual estimates ϵ_k^i output from the timing error detection algorithm in iteration i. The updated timing offset τ^i for iteration i is then found as

$$\begin{aligned} \epsilon^i &= \frac{1}{N}\left(\sum_{k=0}^{N-1}\epsilon_k\right), \\ \hat{\tau}^i &= \hat{\tau}^{i-1} + \beta\epsilon^i. \end{aligned} \tag{10.18}$$

10.10 Gardner Timing Error Detection Algorithm

The Gardner timing estimation algorithm was developed in 1986 [110], and has been widely used in communications systems implementing feedback timing recovery. The Gardner algorithm is non-data-aided, and does not require phase synchronization to operate, but requires two samples per symbol. However, the Nyquist sampling rate for raised cosine pulses with some rolloff requires sampling more than once per symbol anyway.

The timing error estimate ϵ_k produced by the Gardner timing estimation algorithm is given by (similar to the form presented in [111])

$$\epsilon(kT+\tau-\hat{\tau}) = \epsilon_k = \mathrm{Re}\left[\{y^*(kT+\tau-\hat{\tau}) - y^*((k+1)T+\tau-\hat{\tau})\}\, y((k+1/2)T+\tau-\hat{\tau})\right], \tag{10.19}$$

where y is the received sampled sequence, τ is the original timing offset, $\hat{\tau}$ is the current estimated timing offset, so the current remaining timing offset is $\tau - \hat{\tau}$ and the symbol timing T is twice the sampling time T_s. The output timing error estimate ϵ_k is proportional to $\hat{\tau} - \tau$, so a negative ϵ_k indicates that $\tau > \hat{\tau}$, and $\hat{\tau}_k$ should be increased. Conversely, a positive ϵ_k indicates $\hat{\tau} > \tau$, and $\hat{\tau}_k$ should be decreased. Thus β of Equation 10.18 should be negative.

The algorithm operates as follows in a high SNR/low noise environment. When there is no symbol transition from symbol time $k-1$ to k, the difference between the symbol samples will be approximately zero, the half-symbol timing sample at $k-1/2$ will not be counted, and the detector error output is zero. If there is a symbol transition, the half-symbol timing sample will be multiplied by the symbol difference from time $k=1$ to k. If the symbol difference is negative, and the timing offset is positive but less than $T/2$, then for BPSK symbols, the half-symbol timing sample will be positive and the sign of the detector error output will be negative. A negative error output subtracts off of the positive timing offset, moving the resulting timing offset $\tau - \hat{\tau}$ closer to zero.

Conversely, if the timing offset is negative but greater than $-T/2$, the sign of the detector error output will be positive, adding onto the negative timing offset and moving the remaining timing offset $\tau + \hat{\tau}$ closer to zero. For timing offsets between the range of $-T/2$ and $T/2$, a negative timing offset generates a positive error output with symbol transitions, and a positive timing offset generates a negative error output. Both cases drive the remaining timing offset to zero. This will be shown visually in Section 10.11 by examining the S-curve response, which is the output of the Gardner timing estimation algorithm for varying timing offset.

A complexity analysis of the GTED shows that each symbol requires 1 subtraction (add), 1 multiply, 1 conversion to real and 2 conjugations. There is an additional multiplication for each symbol due to the convergence parameter β of Equation 10.17. For k symbols, this requires k additions, $2k$ multiplications, $2k$ conjugations and k conversions to real. To obtain a constant timing estimate $\hat{\tau}$ for the codeword sequence, the timing offset estimates are averaged, resulting in an additional k additions and 1 multiply. The total is $2k$ additions and conjugations, $2k+1$ multiplies and k conversions to real. This algorithm requires 2 samples per symbol for each symbol timing estimate. Therefore, interpolation is over $2k$ samples. Interpolation requires $2kI$ multiplies and additions. Total complexity per iteration is $2k(I+1)$ additions, $2k(I+1)+1$ multiplies, $2k$ conjugations and k conversions to real. Complexity is dominated by the additions and multiplies.

In comparison to the feedforward algorithm presented earlier, which used $k(I+1)D$

multiplies, $k(I+1)D$ adds, kD conjugations and D compares per iteration, the feedback algorithm requires a factor of $D/2$ times fewer multiplications and additions. For any reasonable discretization (say $D \geq 10$), this results in a significant savings in complexity for the feedback algorithm.

The timing estimate ϵ goes through the first-order timing loop filter of Equation 10.18 to generate an estimate $\hat{\tau}$ of the timing offset τ.

This NDA timing estimation algorithm provides an initial timing estimate $\hat{\tau}^0$ and interpolated sampled sequence $\mathbf{y}^0$ for use by the inner D8-PSK APP decoder in its initial iteration, before any symbol estimates are available from the inner decoder. An initial timing estimate should provide better performance for the concatenated system, especially when phase estimation is incorporated into the iterative decoding loop in addition to timing estimation.

Once iterative decoding has begun, however, soft symbol estimates according to Equation 9.4 are available from the inner decoder. These symbol estimates should be used to provide additional information to the timing estimator. A data-directed (DD) timing estimation algorithm is thus preferred for use in the iterative decoding loop.

The Gardner timing estimation algorithm may be converted to DD form by using soft symbol estimates $\tilde{x}_k$ and $\tilde{x}_{k+1}$ in place of the samples $y(kT+\tau-\hat{\tau})$ and $y((k+1)T+\tau-\hat{\tau})$. Because soft symbol estimates $\tilde{x}_k$ are used, which incorporate any existing phase offset, this DD form should perform better without phase synchronization than a DD form using hard symbol decisions $\hat{x}_k$. The DD form with hard symbol decisions $\hat{x}$ is given in [111], pg. 380, as the data-transition tracking loop (DTTL), but requires a phase estimate $\hat{\theta}$.

The output of the DD Gardner timing estimation algorithm is

$$\epsilon(kT) = \epsilon_k = \mathrm{Re}\left[\left\{\tilde{x}_k^* - \tilde{x}_{k+1}^*\right\} y\left((k+1/2)T + \tau - \hat{\tau}\right)\right] \qquad (10.20)$$

Note that for implementation purposes, the Gardner timing error detector (GTED) is the same whether the NDA or DD form is used; only the inputs (samples or soft symbol estimates) change. For this reason, it is practical to use both forms of the algorithm, with the NDA form used initially and the DD form used once iterative decoding begins.

Another commonly used algorithm is the Mueller and Müller (M&M) symbol synchronizer [112] or timing error detector. This algorithm is popular due to the fact that it only requires one sample per symbol, operating at the symbol data rate. As Nyquist sampling requires two samples per symbol for the raised cosine pulse, an algorithm using only one sample per symbol does not save significantly on complexity.

The M&M timing error detector is also a DD feedback timing estimation algorithm, and so can use the soft symbol estimates $\tilde{\mathbf{x}}$ available from the inner D8-PSK decoder. However, the M&M TED does not have a NDA form, and requires phase synchronization to operate well. Thus the M&M TED is not well suited for the combined APP phase and timing estimation, nor can it provided an initial timing estimate to aid the iterative decoding process.

The M&M timing estimation algorithm produces a timing error signal of the form

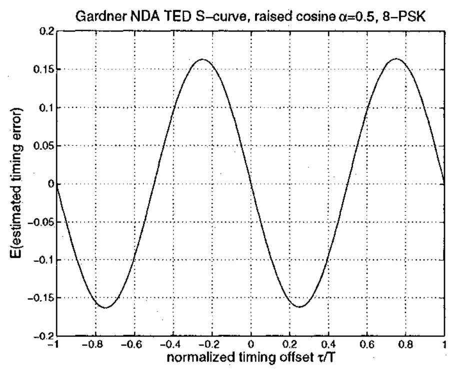
$$\epsilon_k = \epsilon(kT + \hat{\tau}) = \mathrm{Re}\left[\hat{x}^*_{k-1} y\left(kT + \tau - \hat{\tau}\right) - \hat{x}^*_k y\left((k-1)T + \tau - \hat{\tau}\right)\right]. \tag{10.21}$$

10.11 Timing Error Detector Output S-Curve

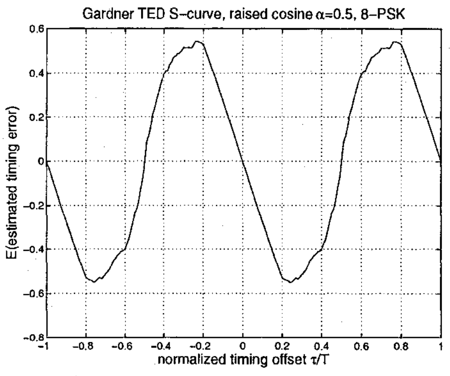
The timing error detector S-curve displays the output $E(\epsilon)$, the expectation of the timing error ϵ, of the timing error detector on the y-axis for a given timing offset τ along the x-axis. The S-curve is an instructive visual representation of the timing estimation algorithm response. The timing detector output displays a sideways S sinusoidal or sawtooth response, and should be zero for no timing offset. Figure 10.4A displays the S-curve for the NDA Gardner TED, and Figure 10.4B shows the DD Gardner TED's S-curve. Figure 10.4C presents the DD M&M TED S-curve. Both timing offset and timing error are normalized by dividing by the symbol timing T. The expectation $E(\epsilon)$ of the timing error is used by taking the mean of the TED output ϵ for each symbol. A sampling rate of two samples per symbol was used for the GTEDs and a sampling rate of one sample per symbol for the M&M TED, for a sequence of 1000 total symbols.

Uncoded 8-PSK-modulated symbols and raised cosine pulses with $\alpha = 0.5$ are used for all S-curves. For the data-directed TEDs, the estimated hard decision symbols $\hat{x}$ are chosen as the sign of every second sampled symbol for the Gardner DD TED and every symbol sample for the M&M TED. Soft symbol estimates are not available as only uncoded BPSK is used for these S curves.

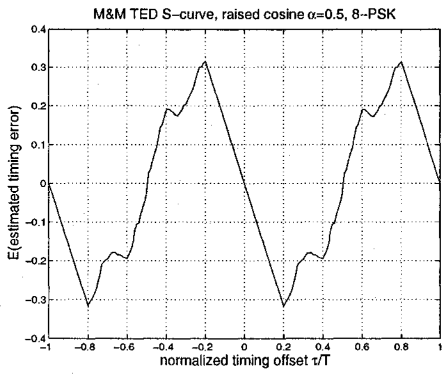
The NDA GTED S-curve is a smooth sinusoid, while the DD GTED and M&MTED S-curves are jagged sawtooth curves. The difference is due to the use of estimated symbols $\hat{x}$ in the DD form. As the timing offset approaches $T/2$, the reliability of the symbol estimates becomes very poor, causing the steep slope to zero.



A



B



C

Figure 10.4: S-curves for non-data-aided and data-directed Gardner timing error detection algorithms (A and B respectively), and the data-directed Mueller and Müller timing error detection algorithm (C), using 8-PSK modulation and raised cosine pulses with $\alpha = 0.5$.

All detectors produce zero output for zero timing offset, generate negative output with a positive timing offset $< T/2$, and generate positive output for a negative timing offset $> -T/2$. The sign of the timing detector's estimated error output e corresponds to the sign of the actual timing error $\hat{\tau} - \tau$ with the timing detector's original estimate of the timing error $\hat{\tau} = 0$. Above $|T/2|$, the sign switches because now the detector is locking onto the next symbol as closest.

At $|T/2|$ and $|T|$, the output of the estimator is also zero. Integer values of T are stable operating points for the timing estimator. The estimator output sign is such as to push the estimate towards the nearest value of T and keep it there. Values offset by $T/2$ from integer values of T are not stable operating points, even though the estimator output is also zero at this point. The estimator output sign near $|T/2|$ offsets is such as to push the estimate away from $|T/2|$ and towards the nearest integer value of T. Near $|T/2|$, the output magnitude is very small, and thus the estimator may take several iterations to move away from the $|T/2|$ point.

Because integer values of T are stable operating points for the timing estimator, the estimator may lock onto a non-zero integer value of T instead of 0. Unfortunately, for a block-coded or turbo-coded system, this symbol shift causes decoder failure because the block or frame is offset. Rather than locating symbol k at time kT, the symbol shift causes symbol $k \pm 1$ to be chosen instead at time kT.

This phenomenon of locking onto a symbol-offset timing estimate is known as **cycle slip**, and is a frame synchronization issue more than a timing synchronization issue. Cycle slips are a common mode of failure in timing recovery for block codes or interleaved codes, and can result in raised error floor and shifted waterfall if not corrected. The effect of cycle slips and methods for mitigation will be discussed further in Section 10.15. A method of correcting cycle slips with a timing offset of $\pm T/2$ is presented in Section 10.15.1. Frame synchronization and its effect on timing offset distribution is discussed later in Section 11.4.

10.12 Interpolation

In addition to the timing estimation algorithm, an interpolator is required to approximate the corrected sampled values $y(kT_s+\tau-\hat{\tau}^i)$, given the original received sampled sequence $y(kT_s+\tau)$ and current estimated timing offset $\hat{\tau}^i$ in iteration i.

Interpolation is possible because the Nyquist-Shannon sampling theorem shows that if a continuous-time signal $s(t)$ is sampled at the Nyquist sampling rate $1/T_s$, then the original signal can be perfectly reconstructed from those samples $s[k] = s(kT_s)$. Therefore, it is also possible to reconstruct samples of the original signal at some other point in time, say $s'[k] = s(kT_s + \tau)$, from the samples $s[k]$. The Nyquist-Shannon interpolation formula states that, when $s[k]$ is sampled at the Nyquist sampling rate $1/T_s > 2B$, any point in $s(t)$ may be reconstructed as

$$s(t) = \sum_{i=-\infty}^{\infty} s[i]\text{sinc}\left(\frac{t-iT_s}{T_s}\right), \tag{10.22}$$

where the normalized sinc function $\text{sinc}(x)$ is defined as $\sin(\pi x)/(\pi x)$.

If the original samples are $y(kT_s+\tau)$, an interpolated version $y(kT_s+\tau-\hat{\tau})$ to correct for the offset τ may be calculated as

$$y(kT_s+\tau-\hat{\tau}) = \sum_{i=-\infty}^{\infty} y(iT_s+\tau)\text{sinc}\left(\frac{(k-i)T_s+\tau-\hat{\tau}}{T_s}\right) \tag{10.23}$$

by substituting $t = kT_s+\tau-\hat{\tau}$, which are the original sampled times with the estimated offset subtracted off, into Equation 10.22.

The ideal interpolating function as shown in Equation 10.22 is a sinc time-domain function of infinite length. In reality, infinitely many samples are not available, so even if a sinc function (rectangular pulse in the frequency domain) is used, it is truncated to use the available samples. Truncation of the sinc function is equivalent to multiplying it with a rectangular window, or convolving with a sinc pulse in the frequency domain. This causes slight degradation as the frequency response is no longer rectangular but has ripples due to Gibbs phenomenon. Windowing functions which provide a smoother frequency response, such as the Kaiser, Hanning or Blackman windows [113] can be used instead.

Note that a significant number of terms are required even for a truncated version of Equation 10.22; the slow die-off of the sinc pulse results in a large number of contributing terms for each new calculated sample $y(kT_s + \tau - \hat{\tau})$.

A much simpler method of interpolation, while not exact, is linear interpolation. Easily implemented, linear interpolation uses only the two adjacent samples $s[k]$ and $s[k+1]$ to calculate any sample located between the two, as

$$s(kT + \mu T) = (1 - \mu)s(kT) + \mu s((k+1)T) \tag{10.24}$$

where μ is a fractional delay. Linear interpolation is simple to implement, but causes performance degradation unless a large number of samples per symbol are used, reducing the distance between each sample so that $s(t)$ is more nearly linear between each sample.

Better performance is obtained with the use of higher-order polynomial interpolation formulas, utilizing more symbol samples in the interpolation formula. An m^{th}-order polynomial interpolating function $P(t)$ uses $m+1$ samples $y_0, \ldots, y_m = y(t_0), \ldots, y(t_m)$ and finds the unique curve of degree m passing through all $m+1$ samples, so that $P(t_i) = y_i, \forall i = 0, \ldots, m$. The polynomial $P(t)$ is then used to find values between the existing discrete samples y_i. Though a high-degree polynomial using many samples might seem to offer increased accuracy, the higher the polynomial degree, the greater the oscillation between the data points.

The Lagrange form of the interpolation polynomial is a linear combination of Lagrange basis polynomials $l_i(t)$, as

$$P(t) = \sum_{i=0}^{m} y_i l_i(t). \tag{10.25}$$

The Lagrange basis polynomials or Lagrange coefficients are found as

$$l_i(t) = \prod_{j=0,\neq i}^{m} \frac{t-t_j}{t_i-t_j} = \frac{t-t_0}{t_i-t_0} \cdots \frac{t-t_m}{t_i-t_m}. \tag{10.26}$$

The first-order Lagrange interpolator is also the linear interpolator of Equation 10.24, as there is only one first-order polynomial which passes through two given points. The cubic Lagrange interpolator, which has four Lagrange coefficients, provides a good compromise between ease of implementation and accuracy. A cubic Lagrange interpolator provided a maximum mean square error (MSE) of 0.0026 with an offset of $0.5T$ and 40% rolloff raised cosine pulse, compared to 0.02 for linear interpolation [114].

A simple realization of the polynomial interpolator, called the Farrow structure [115], was devised in 1988. The Lagrange coefficients are rearranged as a bank of m parallel filters with the output of the ith branch multiplied by μ^i and then summed. The Farrow structure only requires an update of the fractional delay μ, and is thus very efficient for recalculation of a changing delay.

For implementation, use of Lagrange polynomial interpolation with a Farrow structure would be a very practical choice. In this thesis, however, a truncated sinc interpolator was used to generate new discrete samples $y(kT_s+\tau-\hat{\tau})$ in between the original samples $y(kT_s+\tau)$. The sinc interpolator had 128 taps.

10.13 Iterative Timing Recovery

Iterative timing recovery moves the timing estimation block from prior to decoding to within the iterative decoder. Several methods of iterative timing recovery have been presented in the literature. Some methods incorporate timing recovery into the APP decoder, some use a single decoder but iterate between the decoder and timing estimator, and a few use the soft outputs from the decoder to iteratively improve the timing estimate.

Probably the earliest presentation of iterative timing recovery is the application of the expectation-maximization (EM) algorithm to an uncoded BPSK system for a constant timing offset in 1991 [116]. The EM algorithm is iterative, with an expectation step and a maximization step corresponding to one iteration.

A joint ISI-timing trellis using a BCJR-like algorithm to generate soft timing estimates in a manner similar to the channel equalization trellis used for iterative turbo equalization in [117] is presented in [118]. This timing estimation trellis has significant complexity, as the timing trellis based on the discretization of the timing offset has $Q+2$ states and the ISI trellis, representing the ISI due to the modulation pulses with timing offset, has 2^I states for pulse length of I symbols and BPSK modulation. The joint ISI-timing trellis has $2^I(Q+2)$ states.

The joint ISI-timing trellis decoder was combined with a rate 1/4 LDPC decoder, with each iteration of LDPC decoding followed by an iteration of ISI-timing decoding. A Gaussian random walk timing offset model was used with σ_τ/T=0.003. The channel is a partial-response PR4 ($H(D) = 1 - D^2$) channel with $I = 2$. No precoding was used. The joint ISI-timing trellis was quantized with $Q = 5$ and $I = 2$, resulting in 28 states. Note that the trellis complexity is relatively low (on the order of a 32-state or memory 5 convolutional code) due to the short overlap of the partial response channel. For a raised cosine pulse over an AWGN channel, the pulses last significantly longer than 2 symbol periods.

Performance results 0.6 dB from perfect timing were achieved with the joint ISI-timing trellis iterative timing recovery, at a BER of 10^{-4} with 10 decoding iterations. Conventional timing recovery before decoding performed very poorly, with a BER floor near 3×10^{-3}.

A coarse timing estimation algorithm is presented in [119] for turbo decoding, which duplicates the turbo decoder. This method oversamples the received signal by four times, so each sample is spaced $T/4$ from the next sample. A soft metric derived in [120] which is signal- and channel-dependent is found for each of the four sampled sequences. The two samples with the largest metrics are chosen as indicating the sequences which bracket the timing offset on the left and the right. Both chosen sequences are sent to two separate turbo decoders. No further timing estimation is done, but frame synchronization is performed and the sequences shifted by a symbol if required during any iteration. In the final iteration, the LLRs from both decoders is weighted and combined to generate a soft output sequence used for bit decisions.

Good performance is obtained with a rate 1/2 $N = 256$ turbo code using 30% rolloff RRC pulses over an AWGN channel, with approximately 0.1-0.2 dB loss compared to perfect timing. A constant timing offset uniformly distributed from (-T,0) per codeword frame was used.

The complexity increase of this method is in implementation area, rather than latency, as the turbo decoder is duplicated to provide for decoding of two different samples. This method does not require interpolation, which does save on complexity and latency, but that savings is overshadowed by the significant cost of duplicating the turbo decoder. The coarse timing estimate, within a $T/4$ interval, could well show more loss for a longer turbo code.

Both these methods result in considerably increased complexity by including timing estimation within the decoding trellis, or by duplicating the turbo decoder. Methods which incorporate a conventional timing estimation algorithm within the iterative decoding loop add only minimal complexity and latency; they are considered below.

Timing recovery was included with a turbo equalizer for iterative decoding of a partial-response system, consisting of a convolutional-encoded signal separated by an interleaver from a precoder and PR4 pulse shaping [121], [122]. Each iteration of turbo equalization/decoding used LLR estimates from the convolutional decoder as input to a Mueller & Müller TED. With a rate 1/4 convolutional code, results 0.2 dB from turbo equalization with perfect timing were achieved after 25 iterations, and performance 4.7 dB better than a conventional system with timing recovery performed before turbo equalization and decoding at a BER of 2×10^{-5}. A Gaussian random walk timing offset model with $\sigma_\tau/T = 0.003$ was used. The same method was also employed with a rate 8/9 $N = 4096$ LDPC decoder [123]; after 100 iterations, performance was 1 dB from perfect timing and

3 dB better than conventional timing recovery.

This procedure of incorporating timing estimation within the iterative decoding loop, using soft symbol estimates from an APP decoder to drive a data-directed timing algorithm, is basically what we propose for APP timing estimation, but within the iterative turbo decoding loop rather than an equalizer/decoder loop.

A frequency offset estimation-based timing recovery scheme (FOSTR) operating within an LDPC and turbo equalization iterative decoding loop is presented in [124], that uses symbol decisions from the LDPC to generate decision-directed metrics for a feedforward timing estimate. Coherent decoding results are achieved with the FOSTR scheme for a rate 16/17 $N = 4352$ LDPC over an EPR4 partial response channel with a sinusoidally-varying time delay after 30 decoding iterations. Conventional timing estimation using a Mueller & Müller TED loses 0.6 dB in performance at a BER of 10^{-5}.

An approach for LDPC decoding which utilizes the parity check constraints as a metric for finding a coarse timing estimate, and then fine-tuning the estimate with a Mueller & Müller TED was recently presented in [125]. LDPCs are decoded iteratively with the sum-product algorithm which operates on the factor graph of the code's H matrix. Messages are sent back and forth along connecting edges between the codeword bits and parity check constraints, rather than along a trellis. Also, rather than having two component codes, the LDPC has two types of nodes, codeword bits and parity checks, which perform different decoding operations.

This method uses two timing loops, one with a coarse offset step size of $0.2T$ which uses a search criterion (feedforward timing estimation), choosing the estimated timing offset producing the greatest number of satisfied check constraints after three iterations of LDPC decoding. This requires 10 interpolated points for each symbol, compared with one interpolated point per symbol for feedback methods. A frequency offset is also tracked in the same manner. Refinement of the coarse timing estimation is done within the second timing loop, which uses a Mueller & Müller TED operating on the LLRs provided from the LDPC with every iteration.

For a constant timing offset of $\pm T/2$ and 30% rolloff RRC pulses, a rate 1/2 $N = 1944$ LDPC decoded jointly with timing recovery using code constraint feedback produced results 0.2 dB from perfect timing in 20 decoding iterations.

10.14 APP Timing Estimation

The principle of APP timing estimation is the same as APP channel estimation: use the APP probabilities available from the inner D8-PSK APP decoder to generate soft symbol estimates on the D8-PSK symbols. These soft symbol estimates are used instead of hard symbol decisions in a timing estimator.

A data-directed timing estimator, such as the DD form of the Gardner TED or the Mueller & Müller TED, calculates a timing error estimate using the soft symbol estimates. The loop filter generates a new timing offset estimate $\hat{\tau}^i$ with every iteration. An interpolator then uses the received samples y (sampled at $kT + \tau$) and the timing offset estimate $\hat{\tau}^i$ to calculate new sample values $\mathbf{y}^i$ at $kT + \tau - \hat{\tau}^i$. Improved channel metrics $p(\mathbf{y}^i|\mathbf{x})$ are then generated for use by the inner D8-PSK in the next decoding iteration.

The APP timing estimation block is shown in Figure 10.5, incorporated into the (3,2,2)/D8-PSK iterative decoder. The input to the timing estimation block is the APP probabilities $p_{APP}(\mathbf{x}|\mathbf{y}^{i-1})$, and the output from the timing estimation block is the new channel metrics $p(\mathbf{y}^i|\mathbf{x})$ for the next iteration. The timing estimation loop filter is incorporated into the TED block. Input to the TED block is the soft symbol estimate $\tilde{\mathbf{x}}$, and TED output is the timing offset estimate $\hat{\tau}$.

The APP timing estimation procedure can be summarized step-by-step as follows:

1. initially obtain received noisy samples $y(kT_s + \tau)$, input to NDA TED as no symbol estimates are available yet.

2. NDA TED outputs initial timing estimate $\hat{\tau}^0$ to interpolator.

3. Interpolator generates corrected samples $y(kT_s + \tau - \hat{\tau}^0) = \mathbf{y}^0$ and sends them to the inner D8PSK APP decoder for 1st decoding iteration; inner D8PSK APP can

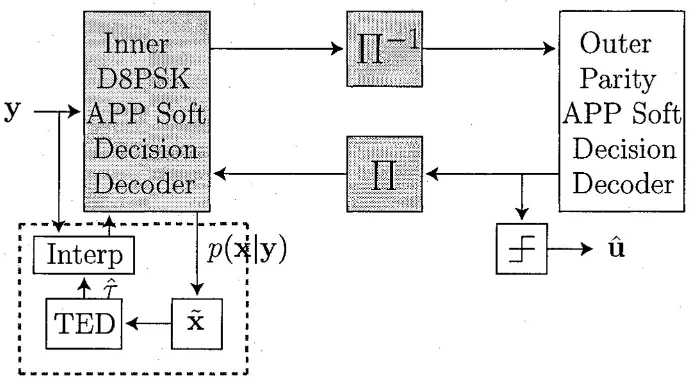

Inner D8PSK APP Soft Decision Decoder
Π^{-1}
Outer Parity APP Soft Decision Decoder
y
Π
Interp
$p(\mathbf{x}|\mathbf{y})$
$\hat{\tau}$
TED
$\tilde{\mathbf{x}}$
$\hat{\mathbf{u}}$

Figure 10.5: Block diagram of iterative decoder with APP timing estimation included.

only use one sample per symbol, so send every other sample.

4. D8PSK APP outputs APP probabilities $p(\mathbf{x}|\mathbf{y}^{i-1})$.

5. Normalized symbol expectation $\tilde{x}_k$ found according to Equations 9.3 and 9.4, input to DD TED along with $\mathbf{y}^{i-1}$.

6. DD TED algorithm generates timing error estimates ϵ_k for each symbol in the codeword block.

7. The timing loop filter averages the timing error estimates and generates an updated timing estimate $\hat{\tau}^i$ according to Equation 10.18.

8. The updated timing estimate $\hat{\tau}^i$ is sent to the interpolator, which generates an interpolated sequence $\mathbf{y}^i \approx \mathbf{y}(\tau - \hat{\tau}^i)$ to correct the timing offset.

9. New channel metrics $p(\mathbf{y}^i|\mathbf{x})$ are calculated as per Equation 5.1, using one sample per symbol. The improved channel metrics are sent to the inner D8-PSK APP decoder for use in iteration $i + 1$.

10. If iteration $i <$ maximum iteration $I_{\max}$, go to step 4 after completing iteration i with outer decoder. If iteration $i = I_{\max}$, complete iteration i, compute $\hat{\mathbf{u}} = \text{sign}(\lambda_{\text{APP}}(\mathbf{u}))$, and stop.

Note that the timing estimation algorithm could stop when the error signal $\epsilon_k = 0$. However, it is possible that the error signal might be zero while the APP probabilities have not yet converged, and in fact with a timing offset near $T/2$, the timing error estimate starts out very near zero. If the APP probabilities change in a future iteration, the error signal might no longer be zero but if timing estimation had ceased, there would be no way to adjust the timing. For this reason, the timing estimation algorithm runs throughout the entire iteration process, though this adds complexity and latency due to both the timing algorithm and the interpolating filter. A practical method to test for convergence would be to see if both the averaged error signal is zero and the mean APP LLR magnitude from the parity code was high, indicating code convergence. If these conditions are met, timing estimation could stop, and iterative decoding could stop a few iterations later. This convergence test was not implemented to stop decoding early with convergence; however, the mean APP (3,2,2) LLR magnitude is later used in Section 10.15 to check for cycle slips of the timing estimator.

10.15 Simulations

APP timing estimation using the Gardner TED was incorporated into the iterative decoding loop for the outer (3,2,2) parity/inner D8-PSK SCC. The NDA form of the GTED was used prior to the decoder to provide an initial timing estimate to boost decoder performance and enable later operation with both phase and timing offset present at the decoder. The DD form of the GTED was implemented within the iterative decoding loop itself. Two samples per symbol were used for both TED algorithms. A sinc interpolation filter with 128 taps was used to generate the corrected samples y^i each iteration.

First, a constant timing offset was assumed constant over not only a codeword frame, but over all codeword frames. The same $S = 3$ interleaver used for simulations in Chapter 9 was also utilized here. Performance results for BER vs SNR with constant timing offsets of $0.4T$ and $0.5T$ are shown in Figure 10.6. Along the waterfall, 50 frame errors were counted for each BER point, while in the error floor, 25 frame errors were counted per

BER point.

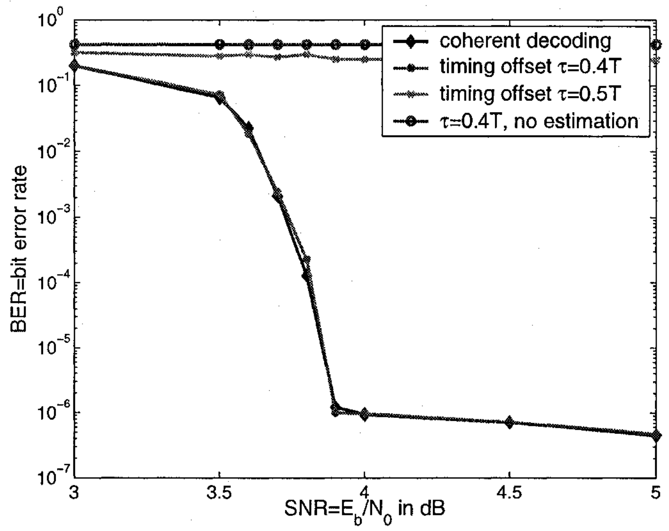


Figure 10.6: BER results for constant timing offsets of $0.4T$ and $0.5T$, with APP timing estimation, compared to coherent decoding over AWGN channel; $S = 3$ 15 kbit S-random interleaver and improved 8-PSK mapping.

Coherent decoding results are achieved for APP timing estimation with a fairly large offset of $0.4T$. Offsets lower than $0.4T$ also achieve coherent performance. An offset of $T/2$ shows very poor performance, however. This is due to decoding failure when the timing estimator has a cycle slip, where it corrects to T instead of 0, which is one symbol off. If the D8-PSK decoder is only one symbol off, the entire codeword will fail, because the interleaver requires perfect symbol positioning to accurately map bits to their appropriate parity codeword. Note that the cycle slip problem is essentially a frame synchronization problem, where a cycle slip shifts the frame positioning off by one or more symbols. A cycle slip occurs approximately 50% of the time, resulting in decoder failure and 50% BER when it does occur. This gives the poor performance of 25% total BER at a timing offset $\tau = T/2$.

A comparison of the BER performance at SNR 4.0 dB with APP timing estimation for a range of timing offset τ is shown in Figure 10.7. The BER drops quickly to coherent decoding results outside of $\tau = T/2$.

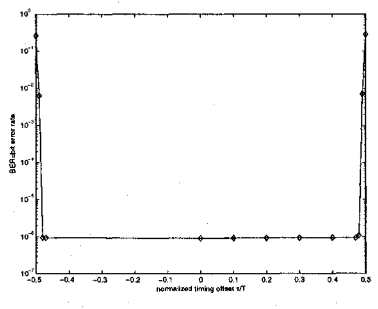


Figure 10.7: BER results at SNR 4.0 dB for a range of constant timing offset τ, using APP timing estimation.

10.15.1 Timing Estimation and Cycle Slips

In a manner similar to the problematic $\pm\pi/8$ phase offset, a timing offset of $\pm T/2$ proves to be the most difficult for the timing estimator. Because the decoder totally fails when the timing estimate is offset by a symbol, the output bit decisions $\hat{u}$ in this case produce 50% bit errors. If the decoder fails, both the APP and extrinsic LLRs from each decoder should be small in magnitude, indicating an inability of the decoder to converge to a solution. A method of detecting a cycle slip is to check the mean LLR magnitude $\mu_{|\lambda(\mathbf{v})|}$ out of the (3,2,2) parity decoder after a number of iterations sufficient to allow for convergence to the correct codeword sequence. If $\mu_{|\lambda(\mathbf{v})|}$ is less than some threshold value in that iteration, then decoder failure can be assumed. Some other methods for detecting cycle slips or frame synchronization in iterative decoders already in the literature use similar metrics [119], [126], [127]. The additional case of slow convergence is not considered, as its effects are overshadowed by cycle slippage. Increasing the number of decoding iterations would help alleviate slow convergence.

Examination of the mean APP (3,2,2) parity LLRs for this coded system show a large differential between LLR values when the decoder converges to the correct codeword and when the decoder does not converge, resulting in errors. The decoder clips output LLRs to $|\lambda_{\max}| = 50$ to prevent numerical overflow, and the mean APP LLR magnitude $\mu_{|\lambda_{\mathrm{APP}}(v')|} = 50$ when the decoder converges to the correct codeword for all SNRs. If the decoder does not converge, $\mu_{|\lambda_{\text{APP}}(v')|}$ is SNR-dependent, but was < 3 for SNR 5 dB and below, and ~ 10 at SNR 10 dB. Therefore, the threshold $\lambda_{\min}$ for assuming decoder failure was set to 20.

A sufficient number of iterations I_{conv} for convergence is also SNR-dependent, but at error-floor SNR values of 4.25 or greater, the decoder converges within 25 iterations. The waterfall SNR values up to 4.0 dB can require all 50 iterations for convergence, but the LLRs reach large magnitude before complete convergence.

Once the cycle slip is detected, the timing offset estimate $\hat{\tau}$ can be changed by $\pm T$, and more decoding iterations performed to allow the decoder to possibly converge using the new timing offset estimate $\hat{\tau} \pm T$. Again, the choice of number of iterations I_{new} required for convergence with the new offset is SNR-dependent but should be at least 20 iterations. The total number of decoding iterations required if a cycle slip is detected is then $I_{\text{conv}} + I_{\text{new}}$. If $I_{\text{conv}} = 30 + I_{\text{new}} = 20$, the total number of decoding iterations equals the original number of decoding iterations $I_{\max} = 50$. Waterfall performance could be impacted with the shortened number of iterations tested for convergence, but $I_{\text{conv}} = 30$ should be sufficient for convergence at error-floor SNRs.

After I_{new} decoding iterations beginning with the new timing offset estimate, the mean (3,2,2) APP LLR magnitudes can be compared with the original (3,2,2) APP LLR magnitudes at I_{conv} to see which estimate provided the best convergence as indicated by the largest APP LLR magnitude. The information bit estimate $\hat{\mathbf{u}}$ corresponding to the timing offset estimate which provided the largest APP LLR magnitude is then chosen.

Note that this method, which is denoted here as APP timing estimation with cycle-slip detection, does not involve any additional complexity or latency over the original APP timing estimation, other than the minimal additional complexity involved in checking for convergence, storing the two mean APP LLR magnitudes and bit estimates at I_{conv} and I_{new}, and comparing the two LLR magnitudes to determine which bit estimate to use after 50 total decoding iterations. Also, if the decoder has converged (or nearly so, as indicated by large LLR magnitude) by I_{conv} iterations, then decoding continues as normal for a total of 50 decoding iterations without adding $\pm T$ to the timing estimate.

10.15.2 Simulation Results with Cycle-Slip Detection

Performance results for the differentially-encoded turbo-coded modulation system with a timing offset of $T/2$ using APP timing estimation with cycle-slip detection are presented in Figure 10.8, compared with the previous $T/2$ timing offset results using APP timing estimation without any cycle-slip detection. Fifty total decoding iterations were used, with $I_{\text{conv}} = 30$, $I_{\text{new}} = 20$ and the mean APP LLR threshold was $\lambda_{\min} = 20$.

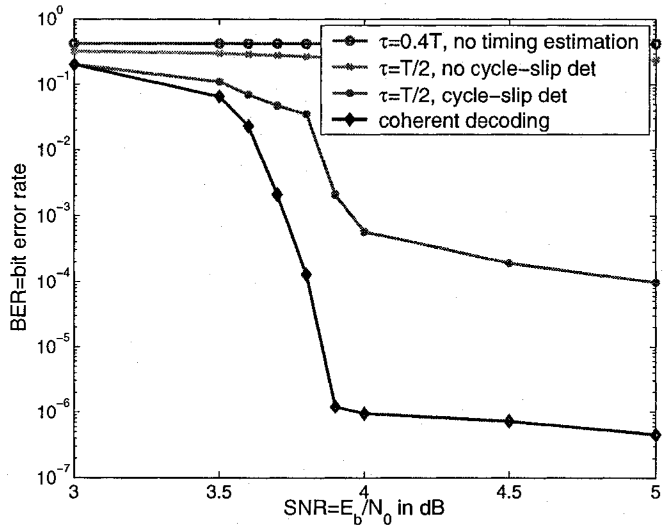


Figure 10.8: BER results for constant timing offset of $T/2$, using APP timing estimation with cycle-slip detection, compared to coherent decoding and APP timing estimation without cycle-slip detection; $S = 3$ 15 kbit S-random interleaver and improved 8-PSK mapping.

The BER performance at $T/2$ is improved significantly, lowering the error floor by three orders of magnitude. However, the error floor is still two orders of magnitude higher for a timing offset of $T/2$ when compared to perfect timing. Increasing the total number of decoding iterations by increasing both I_{conv} and especially I_{new} can help lower this error floor further.

Coherent decoding results are achieved for a timing offset of $0.4T$ with cycle-slip detection as shown in Figure 10.8.

Figure 10.9 displays the BER performance of APP timing estimation with cycle-slip detection at SNR 4.0 for a range of constant timing offsets, from $-T/2$ to $T/2$. In comparison with Figure 10.7, the BER for $|\tau| \geq 0.49T$ is lowered significantly with the use of cycle-slip detection in the APP timing estimation.

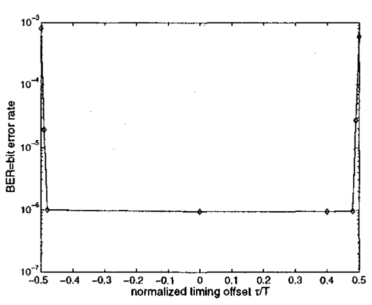


Figure 10.9: BER results at SNR 4.0 dB for a range of constant timing offset τ from $-T/2$ to $T/2$, using APP timing estimation with cycle-slip detection.

A more realistic timing offset model uses a timing offset τ which is constant over one codeword frame, but randomly chosen for each frame, with uniform distribution over $\tau = [-T/2, T/2)$. Clearly, without cycle-slip detection, the higher error floor at $\tau = \pm T/2$ will degrade performance.

Performance results for the uniformly distributed timing offset using APP timing estimation with and without cycle-slip detection are shown in Figure 10.10 for BER vs SNR. The timing offset is constant over a codeword frame, and the channel is AWGN. Fifty decoding iterations are used on each frame, and 25 frame errors were observed for each BER point.

The BER performance is significantly improved by inclusion of cycle-slip detection into the APP timing estimation. Without cycle-slip detection, the error floor is raised by three orders of magnitude compared to coherent decoding, or perfect timing. Inclusion of cycle-slip detection shows near-coherent waterfall performance, although error floor performance is expected to be higher than coherent, due to the raised error floor still

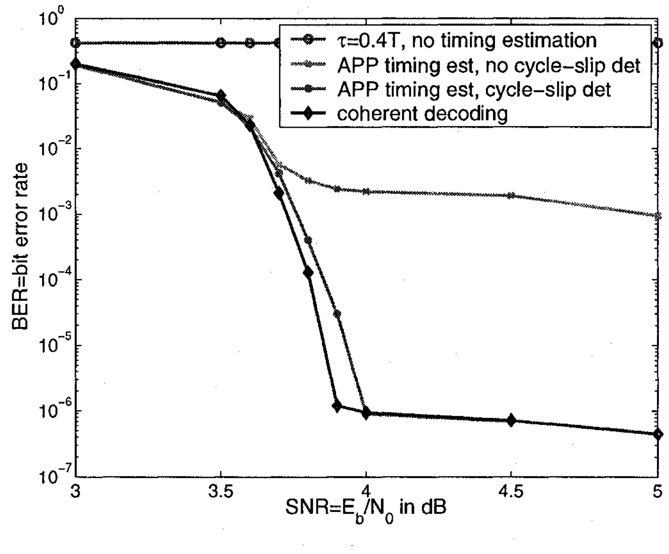


Figure 10.10: BER results for a uniformly distributed constant timing offset τ, using APP timing estimation with and without cycle-slip detection.

existing with $\tau = T/2$. If more iterations are allowed for both I_{conv} and I_{new}, resulting in more total decoding iterations, performance will be improved.

A histogram of the timing offset PDF for the simulation at SNR 4.0 dB is shown in Figure 10.11. A total of 1971 codeword frames were transmitted, resulting in 25 frame errors. The timing offset histogram is approaching a uniform distribution, with calculated $\mu_\tau = -0.007$ and $\sigma^2 = 0.0855$, normalized w.r.t. T. The expected mean and variance for a uniformly distributed variable over the range (-1/2,1/2) are $\mu = 0$ and $\sigma^2 = 0.0833$.

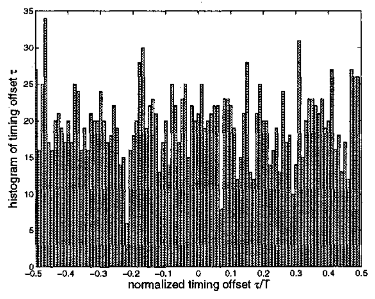


Figure 10.11: Histogram of uniformly distributed constant timing offset τ, for SNR 4.0 dB simulation of Figure 10.10.

Chapter 11

Combined APP Phase and Timing Estimation

This chapter examines the integration of both timing and phase estimation into the iterative decoding process, by incorporating both APP channel and timing estimation, as presented in Chapters 9 and 10.

11.1 Iterative Phase and Timing Recovery

While both phase and timing synchronization have been explored for many years, typically synchronization has been performed before decoding, and considered separately from decoding. Before the widespread use of iterative decoding, this arrangement made sense, as the decoding process had no mechanism to improve the estimation process, and joint phase, timing and data estimation is a very complex problem. With the development of iterative decoding, and iteratively-improving soft information, combined phase and timing estimation and decoding has become feasible. A few recent papers examine joint timing and phase estimation algorithms incorporated into the iterative decoding process.

As mentioned in the previous Chapter 10, [125] combines timing and frequency estimation with sum-product decoding of LDPCs, using the number of satisfied parity check constraints as a metric for choosing a coarse offset estimate based on a search over the discretized timing and phase values. Results presented in [125] are only for separate phase offset or timing offset, not a combined phase and timing offset, but the described algorithm is capable of estimating both, though with significant latency due to a required search over both timing and phase offset.

Several different algorithms are compared in [128] for joint phase and timing synchronization of a turbo code with bit-interleaved coded modulation (BICM), where the turbo code is composed of two rate 1/2 constraint-length 5 convolutional codes and the unpunctured turbo-coded output is mapped to QPSK symbols for a rate 2/3 system with a frame length of 360 symbols. The receiver is assumed to already have estimated the fractional phase and timing error, but a timing offset that is an integer multiple of T and phase offset that is an integer multiple of the M-PSK constellation angle $2\pi/M$ may still exist due to cycle-slippage and constellation phase angle ambiguity. Each technique was tested at the four possible phase constellation angles and for three symbol timing offsets to determine which values maximize the metric of interest, thus requiring twelve combinations. Maximization was performed only for the first iteration, then decoding proceeded using the chosen phase and offset values.

A mode separation (MS) technique proposed in [129] for frame synchronization is examined, which uses the MS or distance between LLR modes $M_+ - M_-$, where $M_+ = E[\lambda|\lambda > 0]$, and $M_- = E[\lambda|\lambda < 0]$ as the metric to be maximized. A pseudo-ML (PML) technique which computes approximate marginal probabilities is also examined, as well as a discrete expectation-maximization (EM) algorithm using soft symbol decisions based on the coded symbol APPs.

Performance results showed 0.2 dB loss in the waterfall region compared to coherent decoding for the discrete EM and PML techniques, and 0.5 dB for MS; all converged to coherent results at high SNR. If 10 iterations of maximization searches were performed for each technique, coherent results were achieved 1 dB earlier than with only 1 iteration of maximization search. A hard-decision-directed EM algorithm was also compared but its performance was extremely poor.

A similar EM algorithm was used in [130] to estimate phase and timing offset as above, for integer constellation angles and symbol times. An unpunctured turbo code with rate 1/2 constraint-length 3 component convolution codes and an interleaver size of 45 bits, with BPSK modulation for a system rate of 1/3. Coherent decoding results were achieved without pilot symbols, while a correlation algorithm with 16 pilot symbols per 135-symbol frame saw 1.5 dB loss at a BER=10^{-3}.

11.2 APP Phase and Timing Estimation

The APP soft information $p(\mathbf{x}|\mathbf{y}^i)$ available from the D8-PSK inner decoder at iteration $i+1$ is used to generate new channel metrics $p(\mathbf{y}^{i+1}|\mathbf{x}, \tilde{\mathbf{h}}^{i+1})$. Updated channel estimates $\tilde{\mathbf{h}}^{i+1}$ are found according to Equations 9.3 to 9.7, and the interpolated sampled values $\mathbf{y}^{i+1}$ found as in Section 10.12.

Both APP channel estimation and timing estimation require a soft symbol estimate $\tilde{\mathbf{x}}$ initially, so the first step in combined APP phase and timing estimation should be to calculate $\tilde{\mathbf{x}}$ according to Equations 9.3 and 9.4. However, which should be computed first, the channel estimate or the timing estimate - or should they both be computed in parallel?

If the timing estimate τ^{i+1} is computed first, then the new interpolated sampled values $\mathbf{y}^{i+1}$ can be used in calculating the instantaneous channel estimates, which become $\hat{\mathbf{h}}^{i+1} = \mathbf{y}^{i+1}\tilde{\mathbf{x}}^*$ as per Equation 9.5. The channel estimate for use in the next iteration, $\hat{\mathbf{h}}^{i+1}$, thus includes the most current interpolated sampled values, $\mathbf{y}^{i+1}$.

However, if the channel estimate $\hat{\mathbf{h}}^{i+1}$ is calculated before the timing estimate τ^{i+1}, then the channel estimate can only use the interpolated sampled values from the previous iteration, $\mathbf{y}^i$. The channel estimate for use in the next iteration is now $\hat{\mathbf{h}}^{i+1} = \mathbf{y}^i\tilde{\mathbf{x}}^*$. It does not contain the yet-to-be-interpolated sampled values $\mathbf{y}^{i+1}$. After the channel estimates are calculated, then the updated timing estimate τ^{i+1} is computed using the soft symbol estimates $\tilde{\mathbf{x}}^*$. New interpolated sampled values $\mathbf{y}^{i+1}$ are then computed, but cannot be used in the channel estimation block until the following iteration, as channel estimation has already been performed.

Determining the channel estimate first essentially decouples the current iteration's channel and timing estimation; both depend on the soft symbol estimates $\tilde{\mathbf{x}}^*$, but not on each other, as the channel estimation must use the previous iteration's sampled values. This method is not using all possible available information for the channel estimation, and is thus inferior to the method which finds the timing estimate first and then the channel estimate.

The combined phase and timing estimation process presented in this thesis calculates the timing estimate τ^{i+1} first, and the new interpolated sampled values $\mathbf{y}^{i+1}$ are then incorporated into the channel estimates $\hat{\mathbf{h}}^{i+1} = \mathbf{y}^{i+1}\tilde{\mathbf{x}}^*$ for use in the D8-PSK APP decoder in the following iteration.

A block diagram of the combined APP phase and timing estimation process is shown in Figure 11.1. The inner and outer decoders and interleavers are not shown in this block diagram, which is a closeup of the estimation procedure. APP probabilities $p(\mathbf{x}|\mathbf{y}^{i-1})$ from the inner D8-PSK decoder are input to the APP phase and timing estimator. Note that $\mathbf{y}^{i-1}$ in the APP probability is the interpolated $\mathbf{y}$ from the previous decoding/estimation iteration $i-1$. The output of the APP phase and timing estimation block are the new channel metrics $p(\mathbf{y}^i|\mathbf{x}, \tilde{\mathbf{h}}^i)$ found in the current iteration i. These updated channel metrics are used as input to the D8-PSK APP decoder during the next decoding/estimation iteration $i+1$. The new channel metrics incorporate both the new interpolated $\mathbf{y}^i$ and the new channel estimate $\tilde{\mathbf{h}}^i$.

The timing estimator assumes a constant timing offset τ over one codeword symbol frame. Thus the loop filter consists of an averaging filter over all the individual symbol timing error estimates ϵ_k output from the timing error detector (TED) for each frame, as described in Chapter 10. With each decoding iteration i, an updated timing estimate τ^i is generated, and a new interpolated $\mathbf{y}^i$.

As mentioned above, the new interpolated $\mathbf{y}^i$ is used as input to the APP phase

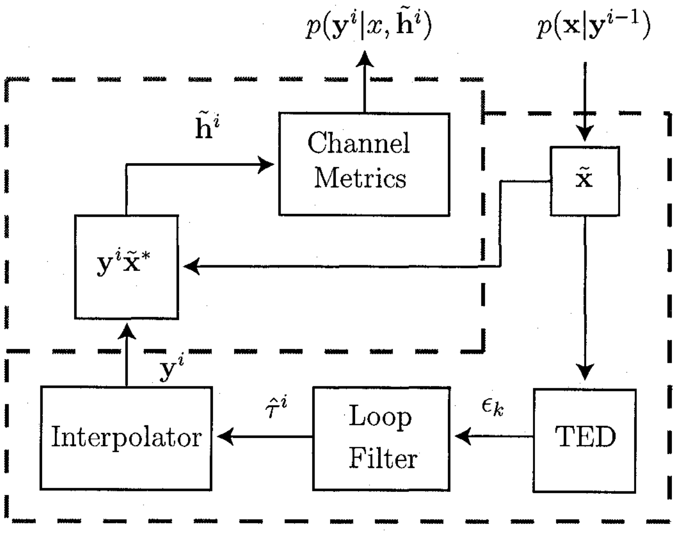


Figure 11.1: Block diagram of combined APP phase and timing estimation.

estimator, which generates new channel metrics for use in the next iteration $i+1$ by the inner D8-PSK APP decoder.

The procedure of combined APP phase and timing estimation is as follows:

1. Before iterative decoding begins, an initial timing estimate $\hat{\tau}^0$ is found from a non-data-aided (NDA) TED, which does not require prior phase synchronization. In this work, the Gardner NDA TED is used. An interpolated received sequence $\mathbf{y}^0 \approx \mathbf{y}(\tau - \hat{\tau}^0)$ is found from the original received sequence $\mathbf{y}(\varphi, \tau)$ with phase offset φ and timing offset τ.

2. For iteration i, the D8-PSK APP decoder outputs APP information on the D8-PSK symbols $\mathbf{x}$ as $p(\mathbf{x}|\mathbf{y}^{i-1})$. For the initial iteration, $i = 1$.

3. APP timing estimator takes APP output on $\mathbf{x}$ from D8-PSK APP decoder as input.

4. Normalized symbol expectation $\tilde{x}_k$ found according to Equations 9.3 and 9.4, input to Gardner DD TED.

5. Gardner DD TED and loop filter generate an updated timing estimate τ^i according to Equation 10.18.

6. The updated timing estimate τ^i is sent to the interpolator, which generates an interpolated sequence $\mathbf{y}^i \approx \mathbf{y}(\tau - \tau^i)$ to correct the timing offset.

7. The new interpolated sequence $\mathbf{y}^i$ is input to the APP channel estimator, along with $\tilde{x}_k$ found in step 4.

8. The APP channel estimator calculates the filtered channel estimates $\tilde{\mathbf{h}}^i$ based on $\tilde{x}_k$ and $\mathbf{y}^i$, as in Chapter 9. New channel metrics $p(\mathbf{y}^i|\mathbf{x}, \tilde{\mathbf{h}}^i)$ are calculated as per Equation 9.7, and sent to the inner D8-PSK APP decoder for use in iteration $i+1$.

9. If iteration $i <$ maximum iteration, go to step 2 after completing iteration i with outer decoder. If iteration $i =$ maximum iteration, complete iteration i, compute $\hat{\mathbf{u}} = \text{sign}(\lambda_{\text{APP}}(\mathbf{u}))$, and stop.

11.3 Simulation Results: Uniformly Random Timing Offset

BER performance results vs SNR were simulated for the differentially-encoded turbo-coded modulation system with outer (3,2,2) parity code and inner differential 8-PSK modulation using the improved 8-PSK mapping of Chapter 6.1 and combined APP phase and timing estimation incorporated into the iterative decoding loop. Performance results are shown in Figure 11.2. An AWGN channel with a constant phase offset of $\pi/16$ radians and a constant timing offset was used. The same $S = 3$ 15 kbit S-random interleaver used in Chapters 9 and 10 was again used here. The convergence parameter $\beta = 0.2$ was used in the timing loop filter, and an averaging filter was used for the phase estimation. One iteration of the APP phase and timing estimation was performed for every decoding iteration.

The curve with asterisks shows the performance of the system with the combined APP phase and timing estimator and a constant timing offset of $0.4T$. Cycle-slip detection is not required for a constant timing offset of $0.4T$. The curve with arrows shows the BER

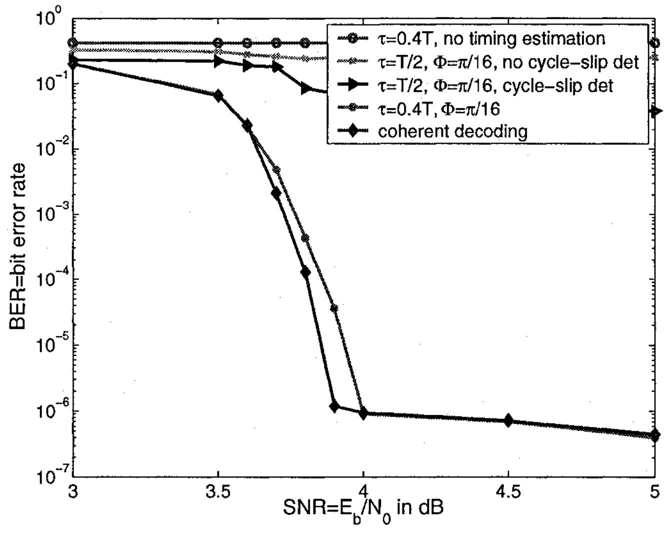


Figure 11.2: BER results with both constant phase offset of $\pi/16$ rads and constant timing offset of $0.4T$ for combined APP phase and timing estimation incorporated into iterative decoding loop; outer (3,2,2) parity code/inner D-8PSK SCC with improved mapping, $S = 3$ 15 kbit S-random interleaver, AWGN channel.

for a constant timing offset of $T/2$ with the combined APP phase and timing estimator with cycle-slip detection. The 'x' curve displays the BER for a constant timing offset of $T/2$ without cycle-slip detection. The top 'o' curve displays the system performance with the phase and timing offsets, but without any phase or timing estimation. Coherent decoding performance is shown in the curve with diamonds for reference. There is a slight 0.1 dB loss in waterfall performance for the combined APP phase and timing estimator with a timing offset of 0.4T, but coherent-decoding results are achieved at higher SNR. Without use of any phase or timing estimation, decoding fails.

Performance results for a constant timing offset of $T/2$ do not give such good results, however. Decoding fails without cycle-slip detection, as for the APP timing estimation in Section 10.15, with a flat BER of 0.25. Not surprisingly, therefore, an offset of $|\tau| = T/2$ leads to a shallow floor from which the algorithm cannot easily escape. Use of cycle-slip detection improves performance somewhat; however, there is still significant degradation in performance for $I_{\text{conv}} = 30$ and I_{new}=20. The combination of both phase and timing offset requires longer convergence times.

Performance results using a uniformly random constant timing offset from $[-T/2, T/2)$ are shown in Figure 11.3 for a phase offset of $\pi/16$ rads. Combined APP phase and timing estimation with cycle-slip detection is incorporated into the iterative decoding loop. Fifty total decoding iterations are used. Twenty-five frame errors are counted for each BER point.

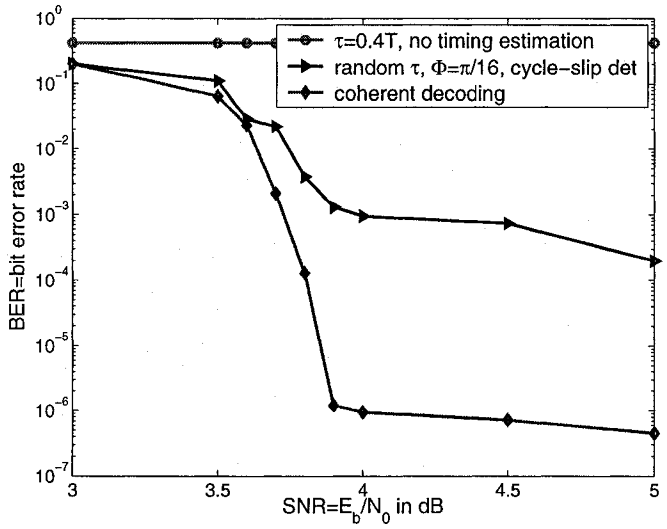


Figure 11.3: BER results with a constant phase offset of $\pi/16$ rads and uniformly random constant timing offset from $[-T/2, T/2)$ for combined APP phase and timing estimation with cycle-slip detection incorporated into iterative decoding loop; outer (3,2,2) parity code/inner D-8PSK SCC with improved mapping, $S = 3$ 15 kbit S-random interleaver, AWGN channel.

A significantly raised error floor exists for the uniformly random timing offset, due to poor performance whenever the offset approaches $\pm T/2$, as indicated in Figure 11.2.

However, is a uniformly distributed random timing offset a reasonable assumption? The distribution of the timing offset depends upon the quality of the frame synchronization prior to sampling and decoding in a bursty packet transmission scenario, and on the initial timing estimation for continuous transmission. Frame synchronization has not been considered thus far in this thesis, and will not be implemented herein, but a brief discussion of frame synchronization and its effect on timing offset is appropriate here.

11.4 Frame Synchronization and Timing Offset

The function of the frame synchronizer is to lock onto a packet and approximately determine the packet's position in time. This can essentially be cast as a detection problem.

The frame synchronizer may detect the packet, or falsely detect noise as being a packet (known as 'false detect'), or fail to detect a packet, resulting in a lost packet. Analysis of false detection and failed detection are beyond the scope of this thesis. The frame synchronizer is assumed to have correctly detected the packet before our synchronization algorithms are applied.

To accomplish packet detection, a header sequence of known symbols is typically sent as the preamble to a frame of data. The receiver knows this header sequence indicates the presence of a frame, and the frame synchronizer looks for the presence of the header sequence. Detection consists of correlating the known header/preamble with the received samples. In packet-based communications, the receiver does not know the packet arrival time, and must correlate successively shifted blocks of samples with the known preamble. A peak in the correlation value is used to indicate the header location.

Multiple samples per symbol are usually taken; those samples nearest the correct timing (with smallest τ) will typically result in the largest correlation. The initial timing clock is then set so as to coincide with the timing of the correlation peak used to trigger detection of the packet.

Assuming that the frame synchronizer correctly detects the incoming packet of data, timing clock initiation starts somewhere within the proper symbol interval, and the timing offset τ is constrained to be within $[-T/2, T/2]$ of the correct sampling time at $T = 0$. But is τ uniformly distributed over that range?

As mentioned above, multiple samples per symbol are typically used in frame synchronization. Thus the sample(s) closest to the actual timing at the pulse peak give the largest correlation, resulting in a bias of the timing offset probability mass towards zero. The timing offset distribution depends on the number of samples per symbol used in the frame synchronization for correlation, as well as the header energy and structure used for frame detection.

More samples per symbol result in finer definition of the pulse peak, and a narrower (lower variance) timing offset probability distribution function (pdf) centered about zero. Thus a uniformly distributed timing offset τ over $[-T/2, T/2]$ does not accurately represent the actual timing offset of the samples after frame detection. The timing offset should be distributed such that the bulk of the probability mass is centered near the optimal sampling time. A Gaussian-distributed timing offset τ is thus proposed as a better model for the actual timing offset.

Some recent work examines the probability density function (pdf) of the timing offset available from the frame synchronizer in a CDMA (code-division multiple-access) spread-spectrum system [131]. For two samples per symbol, a header length of 30 symbols and processing gain of 31, the timing offset pdf is shown to move from an approximately Gaussian shape to a nearly uniform pdf over $[-T/4, T/4]$ with $p(|\tau| > T/4) \rightarrow 0$ as SNR increases.

The probability density function for the timing offset observed in [131] is shown in Figure 11.4 for several SNR values, using two samples per symbol. The SNR given in the figure is $10\log_{10}(E_c/N_0)$, where E_c is the energy per chip, with 31 chips/symbol, a preamble length of 30 bits and BPSK modulation. An E_c/N_0 SNR of -10 dB corresponds to an equivalent SNR$= 10\log_{10}(E_b/N_0) = 10\log_{10}(E_s/N_0)$ of 5 dB. The (3,2,2) parity/D8-PSK concatenated system considered in this thesis has a rate of 2 bits/symbol. For equal normalized symbol energy $E_s = 1$, SNR$= 10\log_{10}(E_b/N_0) = 10\log_{10}(E_s/(RN_0)) = 10\log_{10}(1/N_0) - 10\log_{10}(R)$. Thus for the concatenated system with rate $R = 2$ bits/symbol, an E_c/N_0 SNR of -10 dB corresponds to an E_b/N_0 SNR of 2 dB. This is well below the SNR range of interest for our concatenated system; our system SNR would fall somewhere between $E_c/N_0 = -10$ and -5 dB.

Note that, although the timing offset pdf moves towards a uniform distribution by SNR 0 dB, the distribution is truncated to $[-T/4, T/4]$ rather than $[-T/2, T/2]$ so that the tails at $|\tau| > T/4$ become negligible. This truncated distribution at high SNR is

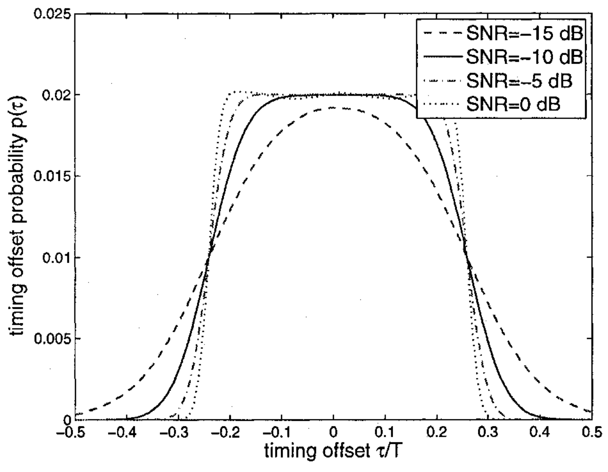


Figure 11.4: Probability density function for timing offset τ obtained from correlation-based frame detector in [131]; shown for several SNR= $10\log_{10}(E_c/N_0)$, with preamble length of 30 bits and 31 chips per symbol.

expected, for the following reason. With two samples per symbol, equal correlation from each sample results (excluding noise) if the samples are spaced an equal distance apart at $\pm T/4$. If the samples are spaced unequally, with one closer than $\pm T/4$ from the peak and the other further away, the closer samples will provide larger correlation. At high SNR, it is extremely likely that the closer samples provide larger correlation than the samples further from the optimal sampling point. Thus the timing offset distribution is truncated at $[-T/4, T/4]$ for higher SNR.

This timing offset distribution has very low probability for $|\tau| > T/4$. The raised error floor observed due to decoding failure in some frames with $|\tau| > .49T$, using a uniformly distributed τ over $[-T/2, T/2]$, would be eliminated with a more realistic non-uniform timing offset distribution.

A plot of the timing offset pdf for SNR= $E_c/N_0 = -10$ dB is shown in Figure 11.5, with the y-axis using a logarithmic scale to better display the probability tails. The tails of the timing offset distribution $p(\tau)$ in the range from $-T/2$ to $-0.49T$ are of most interest to us, as that range is where the timing estimation algorithm occasionally fails due to cycle-slip.

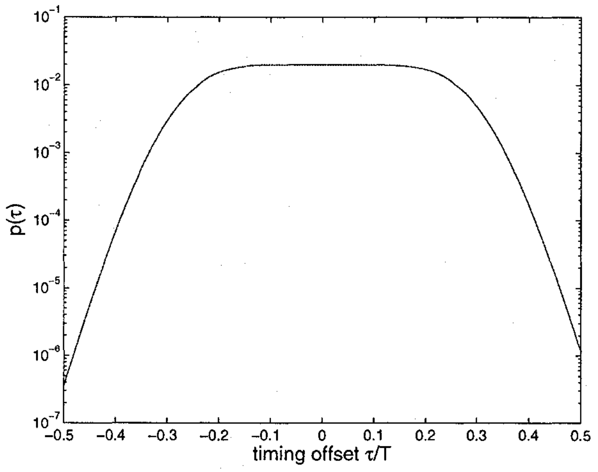


Figure 11.5: Probability density function for timing offset $-T/2 \leq \tau \leq T/2$; obtained from correlation-based frame detector in [131], with SNR= $10\log_{10}(E_c/N_0) = -10$ dB.

From this distribution, an empirical cdf shows that $P(|\tau| \geq 0.49T) = 2.9 \times 10^{-6}$. Thus a potential decoding failure due to cycle slip would only occur once every 350,000 frames. With a constant fixed timing offset τ, the frame error rate at SNR 5 dB ranges from 0.1 for $\tau = T/2$ to 0.006 for $\tau = 0.49T$. An overbound for the estimated BER due to decoding failure at SNR 5 dB can be calculated using the $\tau = T/2$ FER for the entire range of $|\tau| \geq 0.49T$, as $0.1/2 \times 2.9 \times 10^{-6} = 1.5 \times 10^{-7}$. A decoding failure error event is assumed to decode half the bits in the frame incorrectly (which is an overestimation of the actual error). This overestimated BER is still below the coherent decoding floor at SNR 5 dB.

Modelling the pdf of the timing offset with a Gaussian distribution which provides tails greater than those shown in Figure 11.5 in the region of decoder failure, $|\tau| \geq 0.495T$, will provide performance results which serve as a performance overbound.

11.4.1 Doubly-truncated Gaussian Probability Density Function

For simulation purposes, a doubly-truncated Gaussian probability distribution for the timing offset τ is considered, which is constrained to lie within the region $[-T/2, T/2]$ (assuming correct header detection). A doubly-truncated Gaussian pdf is a Gaussian pdf that is constrained to lie within a range limited from below and above, rather than extending to $\pm\infty$ as for a Gaussian pdf.

It will be shown later that a doubly-truncated Gaussian pdf with appropriately-chosen standard deviation σ has larger tails than the timing pdf found in [131]. However, a timing offset with a doubly-truncated Gaussian pdf is simple to generate and simulation results obtained using it will provide an upper bound to performance.

The doubly-truncated Gaussian pdf $p_{\mathrm{DTN}}(x)$ is found from the untruncated Gaussian $p_N(x)$ as [132]

$$\begin{aligned} p_{\mathrm{DTN}}(x) &= \frac{p_N(x)}{\int_A^B p_N(x)dx} = \frac{p_N(x)}{P(x \le B) - P(x \le A)},\ A \le x \le B \qquad (11.1) \\ &= 0 \text{ elsewhere} \end{aligned}$$

where A and B are the lower and upper limits of the doubly-truncated Gaussian, respectively, and $P(x \le B)$ is the Gaussian cumulative distribution function (cdf) $F(x)$ at $x = B$, or $F(B)$.

If the untruncated Gaussian distribution has zero mean and the truncation limits are symmetric about zero, so that the lower limit A equals $-B$, where B is the upper truncation limit, then clearly the doubly-truncated Gaussian also has zero mean. In other words, if $E[x] = \mu = \int_{-\infty}^{\infty} xf(x)dx = 0$ for the untruncated Gaussian, then $E_T[x] = \mu_T = K\int_{-B}^{B} xf(x)dx = 0$ also, where $K = 1/(F(B) - F(-B))$.

If the variance σ^2 of the untruncated Gaussian is such that $p(|x| > B)$ is very small, so the bulk of the probability mass lies inside the truncation limits, then there will be very little probability mass outside the truncation limits that needs to be moved within the allowable limits. In this case, the variance σ_T^2 of the truncated Gaussian will be well-approximated by the variance σ^2 of the untruncated Gaussian distribution.

If a significant amount of the probability mass lies outside the truncation limits, such that $p(|x| > B)$ is not neglectible, then the variance σ_T^2 of the truncated Gaussian deviates from that of the untruncated Gaussian. This case is not considered here, for if $p(|x| > B)$ contributes much to the probability mass, then the truncated Gaussian distribution is not an appropriate pdf model to use.

What variance (or standard deviation σ would be an appropriate choice for the doubly-truncated Gaussian model? For our case, $B = T/2$ and $A = -T/2$, so the doubly-truncated Gaussian is constrained to lie within a region of width T, centered about the mean $\mu = 0$. For a regular Gaussian distribution, 95.45% of the probability mass lies within a distance 2σ of μ, and 99.994% lies within 4σ of μ. Thus if $\sigma = T/8$, only 0.006% of the probability mass of the regular Gaussian would lie outside the limits of the doubly-truncated Gaussian, and the two probability density functions should look very similar.

The cumulative distribution function (cdf) is the probability $F(X) = p(x \leq X)$. The doubly-truncated Gaussian cdf $F_{\mathrm{DTN}}(X)$ is found from the untruncated Gaussian cdf $F(X)$ as [132], for $\mu = 0$,

$$\begin{aligned} F_{\mathrm{DTN}}(X) = P(x \leq X) &= \frac{F(X) - F(A)}{F(B) - F(A)}, \ A \leq X \leq B; \\ &= 0, \ A \geq X; \\ &= 1, \ B \leq X. \end{aligned} \tag{11.2}$$

For $A = -B$, $B = T/2$ and $\sigma = T/m$ with integer m, we obtain the following for the doubly-truncated Gaussian cdf:

$$\begin{aligned} F_{\mathrm{DTN}}(X) = P(\tau \le X) &= \frac{\mathrm{erf}\left(\frac{mX}{T\sqrt{2}}\right) + \mathrm{erf}\left(\frac{m}{2\sqrt{2}}\right)}{2\mathrm{erf}\left(\frac{m}{2\sqrt{2}}\right)}, \quad \frac{-T}{2} \le X \le \frac{T}{2}; \qquad (11.3) \\ &= 0, \ A \ge X; \\ &= 1, \ B \le X, \end{aligned}$$

remembering that $\mathrm{erf}(x)$ is an odd function.

We shall take a closer look at the tail probabilities, by examining the cumulative distribution function of the doubly-truncated Gaussian function. The tail probabilities are expected to be somewhat larger than those for the timing offset distribution model presented in [131]. This is fine, because if a Gaussian-distributed timing offset τ with probability tails greater than Figure 11.5 shows no raised error floor due to decoder failure, the timing offset from the correlation-based frame detector will not generate a raised error floor either.

The tail probability $F_{T/8}(-0.495T) = P(\tau \le -0.495T)$ for $\sigma = T/8$ is found from Equation 11.4 to be $F_{T/8}(-0.495T) = 5.8 \times 10^{-6}$, and thus the double-sided tail probability $P(|\tau| \ge 0.495T) = 1.2 \times 10^{-5}$. Similarly, the tail probability $F_{T/8}(-0.499T) = 1.1 \times 10^{-6}$, and $P(|\tau| \ge 0.499T) = 2.2 \times 10^{-6}$. Comparing with the distribution of [131], the doubly-truncated Gaussian cdf for $\sigma = T/8$ has $P(|\tau| \ge 0.49T) = 2.5 \times 10^{-5}$, while the distribution of [131] at SNR $E_c/N_0 = -10$ dB has $P(|\tau| \ge 0.49T) = 2.9 \times 10^{-6}$. The doubly-truncated Gaussian with $\sigma = T/8$ has a tail probability nearly an order of magnitude greater than the distribution of [131]. However, results in Section 11.5 will show that the doubly-truncated Gaussian timing offset distribution with $\sigma = T/8$ provides near-coherent decoding performance.

Whether the doubly-truncated Gaussian timing offset will cause an error floor for a given σ can be determined as follows. We can estimate the probability of bit error $p_{b,\mathrm{fail}}$ caused by decoding failure from timing offsets near $\pm T/2$ for the concatenated system for any timing offset distribution as

$$\begin{aligned} p_{b,\text{fail}} &\approx \frac{k/2}{k}\int_{\tau=-T/2}^{T/2} p(\text{decoding failure}|\tau)p(\tau)d\tau; \\ &\approx \frac{1}{2}\int_{\tau=-T/2}^{T/2} p(\text{decoding failure}|\tau)p(\tau)d\tau, \end{aligned} \tag{11.4}$$

where k is the information sequence length, if we assume decoding failure consists of half the information bits being decoded incorrectly. This estimate will be an overbound, as not all error events due to cycle slip (failure of timing estimate to converge to correct value) result in 50% error rate.

For timing offset $|\tau| \leq 0.485T$, the probability of decoding failure is negligible. This is shown in Figure 11.6, in a manner similar to the BER results shown in Figure 10.7 in the timing estimation chapter. Figure 11.6 displays the frame error rate (FER) for a range of constant timing offset τ, at SNR 5.0 dB and a constant phase offset $\Phi = \pi/16$ radians. The timing offset τ is positive, ranging from 0 to $T/2$; the FER values are symmetrical about 0, and thus an even function.

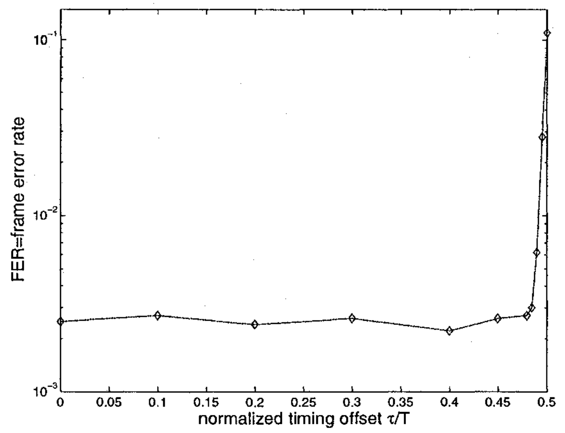


Figure 11.6: FER results at SNR 5.0 dB and constant phase offset $\pi/16$ rads, for a range of constant timing offset τ, using combined APP phase and timing estimation with cycle-slip detection.

The FER rapidly converges to coherent decoding results by $\tau \leq 0.49T$. The FER results include error events which are not due to cycle slip as well; these are indicated by the coherent decoding floor. Also, not all cycle slip error events result in total decoding failure with 50% of the information bits decoded incorrectly. However, as an overbound, this is a reasonable assumption for decoding failure. These FER values may then used as $p(\text{decoding failure}|\tau)$ in Equation 11.4 to determine an overbound estimate for the BER due to cycle slip.

Given that $p(\text{decoding failure}|\tau)$ is an even function, the limits of integration in Equation 11.4 are symmetrical about zero, and $p(\text{decoding failure}|\tau) \approx 0$ for $|\tau| \leq 0.485T$, Equation 11.4 may be rewritten as

$$\begin{aligned} p_{b,\text{fail}} &\approx \int_{\tau=-T/2}^{\tau=0} p(\text{decoding failure}|\tau)p(\tau)d\tau; \\ &\approx \int_{\tau=-T/2}^{\tau=-0.485T} p(\text{decoding failure}|\tau)p(\tau)d\tau, \end{aligned} \tag{11.5}$$

providing that $p(\tau)$ is also an even function. This will be the case for all the timing offset distributions we are considering here.

As discrete values are available from simulation for $p(\text{decoding failure}|\tau)$, it is practical to approximate Equation 11.5 using discrete values of τ. Indicating the error event of decoding failure as df, and assuming $m+1$ values for τ with $\tau_i = -T/2 + iT/200$, we obtain for a fixed SNR

$$\begin{aligned} p_{b,\text{fail}} &\approx \sum_{i=1}^{m} p(\text{df}|\tau_i)p(\tau_{i-1} \leq \tau \leq \tau_i) \\ &\approx p(\text{df}|\tau = -0.495T)p(-T/2 \leq \tau \leq -0.495T) + \ldots \\ &\quad + p(\text{df}|\tau = \tau_m)p(\tau_{m-1} \leq \tau \leq \tau_m) \\ &\approx p(\text{df}|\tau = -.495)F(-.495T) + p(\text{df}|\tau = -.49T)(F(-.49T) - F(-.495T)) + \ldots \\ &\quad + p(\text{df}|\tau = \tau_m)(F(\tau_m) - F(\tau_{m-1})). \end{aligned} \tag{11.6}$$

The incremental increase in τ of $T/200$ matches the available FER data for $p(\text{df}|\tau)$;

obviously this value could be changed to match any increment.

An overbound for the BER due to decoding failure, $p_{b,\text{fail}}$, can then be found from the cdf values of Equation 11.4, the FER values in Figure 11.6 and Equation 11.6. For a doubly-truncated Gaussian distribution with $\sigma = T/8$, an overbound for the BER due to decoding failure at SNR 5 dB is given by $p_{b,\text{fail}} \approx 2.3 \times 10^{-7}$. If σ increases to $T/4$, the overbound for the BER due to decoding failure at SNR 5 dB increases to $p_{b,\text{fail}} \approx 4.5 \times 10^{-5}$.

If the bit error probability due to decoding failure is less than the coherent decoding BER at that SNR, then no raised error floor above that found with coherent decoding should occur with the timing offset pdf of given σ. However, if $p_{b,\text{fail}}$ is greater than the coherent decoding BER, then a raised error floor due to decoding failure when $|\tau| \geq 0.49T$ is likely. Equation 11.6 is an overbound, and could be off by a factor of 2 or more from the actual BER due to decoding failure.

For the doubly-truncated Gaussian timing offset pdf with $\sigma = T/8$, no raised error floor is expected, as coherent decoding results showed a BER of 4.7×10^{-7} at SNR 5 dB; the estimated $p_{b,\text{fail}}$ is about half this, and is an overbound. The actual BER due to decoding failure should be lower than the estimate, and thus should have minimal impact on the coherent decoding curve. When σ increases to $T/4$, however, the estimated $p_{b,\text{fail}}$ increases to the level of a significant error floor, two orders of magnitude higher than that of coherent decoding at SNR 5 dB.

These estimated bit error probabilities due to decoding failure for $|\tau|$ near $T/2$ overbound quite closely the actual BER observed for both σ values in the simulation results of the following Section 11.5.

11.5 Simulation Results: Truncated Gaussian Random Timing Offset

BER performance results vs SNR were simulated for the (3,2,2) parity/D8-PSK concatenated system using the improved mapping of Chapter 6.1, with combined APP phase and timing estimation incorporated into the iterative decoding loop. Figure 11.7 shows BER performance results over an AWGN channel with constant phase offset of $\pi/16$ radians and a doubly-truncated Gaussian random timing offset τ constrained to be within $[-T/2, T/2]$. Both $\sigma = T/4$ and $\sigma = T/8$ random timing offsets are shown. Fifty total decoding iterations are used, with cycle-slip detection and $I_{\text{conv}} = 30$ and $I_{\text{new}} = 20$.

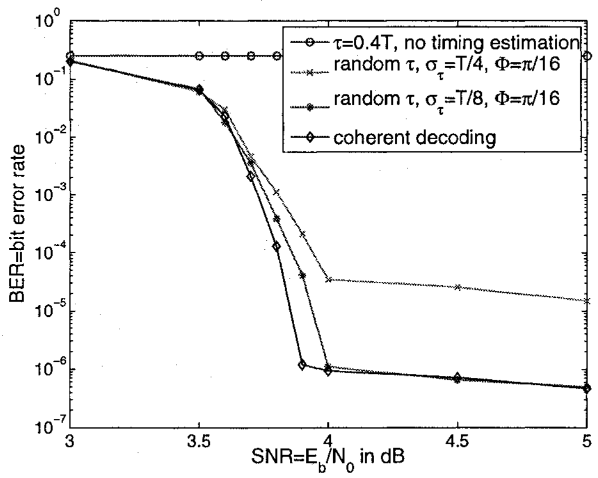


Figure 11.7: BER results with a constant phase offset of $\pi/16$ rads and doubly-truncated Gaussian random constant timing offset from $[-T/2, T/2)$ with both $\sigma = T/8$ and $T/4$, for combined APP phase and timing estimation with cycle-slip detection incorporated into iterative decoding loop; outer (3,2,2) parity code/inner D-8PSK SCC with improved mapping, $S = 3$ 15 kbit S-random interleaver, AWGN channel.

Results show that there is no raised error floor for the doubly-truncated Gaussian random timing offset with $\sigma = T/8$. For $\sigma = T/4$, a significant error floor exists, near 2×10^{-5} at SNR 5 dB. This error floor is below that predicted by the overbound of Equation 11.6 by a factor of 2.

There is a slight loss in waterfall performance for $\sigma = T/8$ of 0.1 dB. At higher SNR, coherent performance results are achieved with $\sigma = T/8$.

When the timing offset model better approximates the actual timing offset distribution out of the frame synchronizer, the performance of the combined iterative APP phase-timing estimation improves dramatically, to near-coherent decoding results.

Chapter 12

Conclusions

A low-complexity serially concatenated system presented in this thesis that combines an outer (3,2,2) parity check code with an inner differential 8-PSK modulation code offers near-capacity performance when iteratively decoding according to turbo principles. This system also provides a relatively high throughput of 2 bits/symbol, due to the higher-order 8-PSK modulation. The rotationally-invariant property of the inner code aids in channel phase estimation, and supplies sufficient soft information from the inner APP decoder to allow initial operation under unknown channel phase rotations.

Both encoding and decoding portions of this system may be implemented with low complexity, and could be used in conjunction with packet transmission, where short messages increase the need for phase offset immunity. Due to a very low MSED, this simple code has a significant error floor. An improved 8-PSK mapping lowers the error floor at a slight 0.2 dB SNR penalty in the turbo cliff onset region.

Significant lowering of the error floor by nearly two orders of magnitude was achieved by development of a short-cycle-free code-matched interleaver. This interleaver design eliminated the low MSED error events possible in the D8-PSK trellis by viewing them as short cycles in the interleaver factor graph.

A low-complexity channel estimation procedure using this soft information from the inner APP decoder achieves near-coherent performance without channel phase information, under both constant and time-varying simulated phase processes. Neither pilot symbols nor differential demodulation were used or required. The random walk phase process channel estimation results in a loss in turbo cliff performance of about 0.25 dB, while the linear phase process results in near-coherent results for $\Delta fT = 4.5 \times 10^{-5}$ to a 0.5 dB loss in turbo cliff for more severe $\Delta fT = 2.1 \times 10^{-4}$. A raised error floor occurs for larger ΔfT due to cycle slips.

APP timing estimation similarly utilizes the APP information from the inner D8-PSK APP decoder to generate soft D8-PSK symbol estimates. These soft symbol estimates are used in a data-directed timing error detector to find an iteratively-improving timing offset estimate. An interpolator then supplies an interpolated received sequence sampled at times with the timing offset estimate removed. New channel metrics for the inner APP decoder are found using the new interpolated sequence.

Incorporation of cycle-slip detection based on the mean parity APP LLRs allows correction of cycle slips by the timing estimation algorithm. Performance is significantly improved for a uniformly random timing offset when cycle-slip detection is included with APP timing estimation, compared to no cycle-slip detection. Coherent decoding results were achieved with a timing offset τ uniformly distributed over $[-T/2, T/2)$ and constant over a codeword frame with APP timing estimation and cycle-slip detection.

APP channel and timing estimation are next combined together within the iterative decoding loop. First, a timing estimate found from a DD TED with soft symbol estimates is used to obtain a new interpolated sequence, sampled with the estimated timing offset removed. The APP channel estimate is found from the interpolated sequence and soft symbol estimates. New channel metrics are obtained using the updated channel estimate which incorporates the latest timing estimate. These improved channel metrics are then used by the inner D8-PSK decoder in the next decoding iteration.

The decoder with combined APP channel and timing estimation integrates both decoding and synchronization into the iterative decoding loop, and allows for packet synchronization without loss of poorly-synchronized initial bits. No training sequences or pilot symbols are required for synchronization, with their corresponding rate loss. Differential demodulation is not required either for channel synchronization.

Coherent decoding results were achieved for a phase offset of $\pi/16$ rads and constant timing offset of $0.4T$. Due to significant performance degradation with a constant timing offset of $T/2$ and the phase offset, even with cycle-slip detection, APP timing estimation for a uniformly distributed timing offset and $\pi/16$ radians phase offset saw 1 dB loss in waterfall performance and a raised error floor even with cycle-slip detection.

However, a uniformly distributed timing offset does not accurately model the timing offset distribution available from the frame synchronizer. A doubly-truncated Gaussian distribution better models the frame synchronizer's ability to locate the symbol timing by the correlation peak. Simulation results with a doubly-truncated Gaussian-distributed constant timing offset showed that near-coherent decoding results could be achieved for $\sigma = T/8$ and $\pi/16$ radians phase offset, with only 0.1 dB loss in waterfall performance and no raised error floor.

In conclusion, this thesis presented a capacity-approaching iteratively-decodable coding system that integrates decoding and synchronization together. The iterative nature of the decoder allows for steadily improving symbol estimates, which in turn provide improved timing and phase estimates, resulting in improved channel metrics for the following iteration. The APP phase and timing estimation procedure is easily implemented, and does not require an exhaustive search over both phase and timing parameters. Near-coherent error-rate results are achieved even in the presence of a combined phase and timing offset.

Bibliography

[1] C. Berrou, A. Glavieux and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo codes," *Proc. IEEE Intl. Conf. Communications (ICC) 1993*, Geneva, Switzerland, 1993, pp. 1064-1070.

[2] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Trans. Communications*, vol. 44, no. 10, pp. 1261-1271, Oct. 1996.

[3] S. Benedetto, D. Divsalar, G. Montorsi and F. Pollara, "Serial concatenation of interleaved codes: Performance analysis, design, and iterative decoding," *IEEE Trans. Information Theory*, 44(3), pp. 909-926, May 1998.

[4] S. ten Brink, "Design of serially concatenated codes based on iterative decoding convergence," in *2nd Intl. Symp. Turbo Codes & Related Topics*, Brest, France, 2000, pp. 319-322.

[5] S. ten Brink, "Convergence behavior of iteratively decoded parallel concatenated codes," *IEEE Trans. Communications*, 49(10), pp. 1627-1737, Oct. 2001.

[6] S. Howard and C. Schlegel, "Differentially-Encoded Turbo Coded Modulation with APP Channel Estimation," *Proc. IEEE Global Telecommunications Conf. (GlobeCom) 2003*, San Francisco, CA, December 1-5, 2003, pp. 1761-1765.

[7] S. Howard and C. Schlegel, "Differential Turbo Coded Modulation with APP Channel Estimation," *IEEE Trans. Communications*, in press.

[8] G. Ungerböck, "Channel Coding with Multilevel/Phase Signals," *IEEE Trans. Information Theory*, vol. 28, pp. 55-67, Jan. 1982.

[9] R.G. Gallager, *Information Theory and Reliable Communications*, New York, Wiley, 1968.

[10] A.J. Viterbi, "Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm," *IEEE Trans. Information Theory*, vol. 13, pp. 260-269, April 1967.

[11] C. E. Shannon, "A Mathematical Theory of Communication," *Bell System Tech. J.*, vol. 27, pp. 379-423 and 623-656, July and October 1948.

[12] J.M. Wozencraft and I.M. Jacobs, *Principles of Communication Engineering*, Wiley, New York, 1965; reprinted by Waveland Press, 1993.

[13] R.G. Gallager, "Low-density parity-check codes," *IRE Trans. Information Theory*, vol. 8, no. 1, pp. 21-28, Jan. 1962.

[14] R.G. Gallager, *Low-Density Parity-Check Codes*, MIT Press, Cambridge, MA, 1963.

[15] D.J.C. MacKay and R.M. Neal, "Near Shannon limit performance of low density parity check codes," *IEE Electronic Letters*, vol. 32, no. 18, pp. 1645-1646, August 1996.

[16] R. W. Hamming, "Error Detecting and Error Correcting Codes," *Bell System Tech. J.*, vol. 29, pp. 147-160, April 1950.

[17] M. J. E. Golay, "Notes on Digital Coding," *Proc. IRE*, vol. 37, p. 657, 1949.

[18] P. Elias, "Error-Free Coding," *IRE Convention Record*, vol. 3, pt. 4, pp. 37-46, 1955.

[19] J.M. Wozencraft and B. Reiffen, *Sequential Decoding*, MIT Press, Cambridge, MA, 1961.

[20] S. Lin and D.J. Costello, *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1983.

[21] C. Schlegel, *Trellis Coding*, IEEE Press, Piscataway, NJ, 1997.

[22] I. S. Reed and G. Solomon, "Polynomial Codes Over Certain Finite Fields," *SIAM J.*, vol. 8, pp. 300-304, June 1960.

[23] G. Ungerböck, "Trellis-coded modulation with redundant signal sets, Part I: Intro-

duction," *IEEE Communications Mag.*, vol. 25, no. 2, pp. 5-11, Feb. 1987.

[24] G. Ungerböck, "Trellis-coded modulation with redundant signal sets, Part II: State of the art," *IEEE Communications Mag.*, vol. 25, no. 2, pp. 12-21, Feb. 1987.

[25] J.M. Wozencraft and R.S. Kennedy, "Modulation and Demodulation for Probabilistic Decoding," *IEEE Trans. Information Theory*, vol. 12, no. 3, pp. 291-297, 1966.

[26] J.L. Massey, "Coding and Modulation in Digital Communications," *Proc. Intl. Zürich Sem. Digital Communications*, Zürich, Switzerland, pp. E2(1)-E2(4), March 1974.

[27] CCITT Recommendations V.34.

[28] U. Black, *The V Series Recommendations, Protocols for Data Communications Over the Telephone Network*, McGraw-Hill, New York, 1991.

[29] R. C. Bose and D. K. Ray-Chaudhuri, "On a Class of Error-Correcting Binary Group Codes," *Inform. Control*, vol. 3, pp. 68-79, March 1960.

[30] A. Hocquenghem, "Codes Correcteurs d'Erreurs," *Chiffres*, vol. 2, pp. 147-156, 1959.

[31] B. Sklar, *Digital Communications: Fundamentals and Applications*, Prentice-Hall, Englewood Cliffs, New Jersey, 1988.

[32] J.L. Massey and D.J. Costello, Jr., "Nonsystematic convolutional codes for sequential decoding in space applications," *IEEE Trans. Communications*, vol. 19, pp. 806-813, 1971.

[33] O.M. Collins, "The subtleties and intricacies of building a constraint length 15 convolutional decoder," *IEEE Trans. Communications*, vol. 40, pp. 1810-1819, 1992.

[34] T.J. Richardson, M. Shokrollahi, and R. Urbanke, "Design of capacity-approaching irregular low-density parity-check codes," *IEEE Trans. Information Theory*, pp. 619-637, Feb. 2001.

[35] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Analysis of low density codes and improved designs using irregular graphs," *Proc. 30th Annual ACM Symp. Theory Computing*, pp. 249-258, 1998.

[36] M. Luby, M. Mitzenmacher, A. Shokrollahi, and D. Spielman, "Improved low-density parity-check codes using irregular graphs," *IEEE Trans. Information Theory,* vol. 47, no. 2, pp. 585-598, 2001.

[37] S.Y. Chung, G.D. Forney, T.J. Richardson and R. Urbanke, "On the design of low-density parity-check codes within 0.0045 dB of the Shannon limit," *IEEE Communications Letters*, vol. 5, no. 2, pp. 58-60, Feb. 2001.

[38] C. Schlegel and L. Perez, *Trellis and Turbo Coding*, IEEE/Wiley 2004, ISBN 0-471-22755-2.

[39] L.R. Bahl, J. Cocke, F. Jelinek and J. Raviv, "Optimal Decoding of Linear Codes for Minimizing Symbol Error Rate," *IEEE Trans. Information Theory*, vol. 20, pp. 284-287, March 1974.

[40] S.S. Pietrobon, G. Ungerböck, L.C. Pérez and D.J. Costello, Jr., "Rotationally invariant nonlinear trellis codes for two-dimensional modulation," *IEEE Trans. Inform. Theory*, vol. 40, pp. 1773-1791, Nov. 1994.

[41] K. R. Narayanan and G. L. Stüber, "A Serial Concatenation Approach to Iterative Demodulation and Decoding," *IEEE Trans. Communications*, vol. 47, no. 7, pp. 956-961, July 1999.

[42] M. Peleg, I. Sason, S. Shamai and A. Elia, "On interleaved, differentially encoded convolutional codes," *IEEE Trans. Information Theory*, 45(7), pp. 2572-2582, Nov. 1999.

[43] D. Divsalar, H. Jin and R. J. McEliece, "Coding theorems for 'turbo-like' codes," *Proc.* 36th *Allerton Conf. on Communication, Control & Computing*, Allerton, IL, USA, pp. 201-210, Sept. 1998.

[44] L.C. Pérez, J. Seghers and D.J. Costello, Jr., "A distance spectrum interpretation of turbo codes," *IEEE Trans. Information Theory*, vol. 42, pp. 1698-1709, Part I, Nov. 1996.

[45] S. Benedetto and G. Montorsi, "Unveiling Turbo-Codes: Some Results on Parallel

Concatenated Coding Schemes," *IEEE Trans. Information Theory*, vol. 43, no. 2, pp. 409-428, March 1996.

[46] J. Tan and G.L. Stüber, "Analysis and design of interleaver mappings for iteratively decoded BICM," *IEEE Intl. Conf. Communications (ICC) 2002*, New York, NY, USA, April 28-May 2, vol. 3, pp. 1403-1407.

[47] S. Howard, C. Schlegel, L. Pérez, F. Jiang, "Differential Turbo Coded Modulation over Unsynchronized Channels," *Proc. IASTED* 3rd *Intl. Conf. Wireless and Optical Communications (WOC) 2002*, Banff, Alberta, Canada, 2002, pp. 96-101.

[48] D. Divsalar and F. Pollara, "Multiple turbo codes for deep-space communications," *JPL TDA Progress Report 42-121*, Jet Propulsion Laboratory, California Institute of Technology, May 1995, pp. 66-77.

[49] D. Divsalar and F. Pollara, "Turbo Codes for PCS Applications," *Proc. IEEE Intl. Conf. Communications (ICC) 1995*, Seattle, WA, June 1995, pp. 54-59.

[50] F. Schreckenbach, N. Görtz, J. Hagenauer and G. Bauch, "Optimized Symbol Mappings for Bit-Interleaved Coded Modulation with Iterative Decoding," *IEEE Global Telecommunications Conf. (GlobeCom) 2003*, San Francisco, CA, USA, Dec. 1-5, 2003.

[51] X. Li, A. Chindapol and J.A. Ritcey, "Bit-Interleaved Coded Modulation With Iterative Decoding and 8PSK Signaling," *IEEE Trans. Communications*, vol. 50, pp. 1250-1257, Aug. 2002.

[52] S. Howard, "Probability of Random Interleaver Containing d_H=2,4 Two-Branch Error Patterns for the (3,2,2) Parity/Differential 8PSK Serially-Concatenated Code," unpublished report, HCDC Laboratory, Dept. of Electrical and Computer Engineering, University of Alberta, December 2003.

[53] S. ten Brink, "*Design of Concatenated Coding Schemes based on Iterative Decoding Convergence*," Ph.D. dissertation, Universität Stuttgart, Germany, Shaker Verlag, 2002.

[54] T.M. Cover and J.A. Thomas, *Elements of Information Theory*, Wiley Series in Telecommunications, John Wiley & Sons, 1991.

[55] S. ten Brink, G. Kramer and A. Ashikhmin, "Design of Low-Density Parity-Check Codes for Multi-Antenna Modulation and Detection," *IEEE Trans. Communications*, vol. 52, pp. 670-678, April 2004.

[56] D. Divsalar and F. Pollara, "Turbo Trellis Coded Modulation with Iterative Decoding for Mobile Satellite Communications," *IMSC 97*, June 97. Available at `http://www331.jpl.nasa.gov/public/tcodes-bib.html`.

[57] S. Crozier and P. Guinard, "Distance Upper Bounds and True Minimum Distance Results for Turbo-Codes Designed with DRP Interleavers," *Proc. 3rd Intl. Symp. Turbo Codes & Related Topics*, Brest, France, Sept. 1-5, 2003, pp. 169-172.

[58] J. Yuan, B. Vucetic and W. Feng, "Combined Turbo Codes and Interleaver Design," *IEEE Trans. Communications*, vol. 47, no. 4, pp. 484-487, April 1999.

[59] W. Feng, J. Yuan and B. Vucetic, "A Code-Matched Interleaver Design for Turbo Codes," *IEEE Trans. Communications*, vol. 50, no. 6, pp. 926-937, June 2002.

[60] M. Oberg, A. Vityaev and P.H. Siegel, "The effect of puncturing in turbo encoders," *Proc. Intl. Symp. Turbo Codes & Related Topics*, Brest, France, Sept. 1997, pp. 184-187.

[61] T.V. Souvignier, M. Oberg, P.H. Siegel, R.E. Swanson and J.K. Wolf, "Turbo decoding for partial response channels," *IEEE Trans. Communications*, vol. 48, pp. 1297-1308, Aug. 2000.

[62] N. Ehtiati, A. Ghrayeb and M. R. Soleymani, "Improved Interleaver Design for Turbo Coded Intersymbol Interference Channels," *Proc. of IEEE Vehicular Technology Conf. (VTC)-2003/Fall*, Orlando, FL, Oct. 6-9, 2003.

[63] C. Fragouli and R. D. Wesel, "Semi-Random Interleaver Design Criteria," *Proc. Comm. Theory Symp. at IEEE Global Telecommunications Conf. (GlobeCom) 1999*, vol. 5, Rio de Janeiro, Brazil, Dec. 5-9, 1999, pp. 2352-2356.

[64] N. Wiberg, *Codes and Decoding on General Graphs.* PhD thesis, Linköping University, Sweden, 1996.

[65] F.R. Kschischang, B. Frey and H.-A. Loeliger, "Factor Graphs and the Sum-Product Algorithm," *IEEE Trans. Information Theory*, vol. 47, no. 2, pp. 498-519, Feb. 2001.

[66] P. Vontobel, "On the Construction of Turbo Code Interleavers Based on Graphs with Large Girth," *Proc. IEEE Intl. Conf. Communications (ICC) 2002*, New York, NY, April 28-May 2, 2002.

[67] R. Garello, F. Chiaraluce, P. Pierleoni, M. Scaloni and S. Benedetto, "On Error Floor and Free Distance of Turbo Codes," *Proc. IEEE Intl. Conf. Communications (ICC) 2001*, vol. 1, Helsinki, Finland, June 11-14, 2001, pp. 45-49.

[68] A. Huebner, D.V. Truhachev and K.Sh. Zigangirov, "On Permutor Designs Based on Serially Concatenated Convolutional Codes," *IEEE Trans. Communications*, vol. 52, no. 9, pp. 1494-1503, Sept. 2004.

[69] J.P. Costas, "Synchronous Communications," *Proc. IRE*, vol. 44, pp. 1713-1718, Dec. 1956.

[70] S.A. Butman, J.R. Lesh, "The Effects of Bandpass Limiters on n-Phase Tracking Systems," *IEEE Trans. Information Theory*, vol. 25, pp. 569-576, June 1977.

[71] H. Meyr and G. Ascheid, *Synchronization in Digital Communications, Vol. 1*, Wiley Series in Telecommunications, 1990.

[72] F.M. Gardner, *Phaselock Techniques*, 2nd edition, John Wiley & Sons, 1979.

[73] R.E. Ziemer and R.L. Peterson, *Introduction to Digital Communication*, 2nd ed., Prentice-Hall, NJ, 2001.

[74] J.G. Proakis, *Digital Communications*, 4th edition, McGraw-Hill, 2000.

[75] E.K. Hall and S.G. Wilson, "Turbo codes for noncoherent channels," *Proc. IEEE Global Telecommunications Conf. (GlobeCom) '97*, Nov. 1997, pp. 66-70.

[76] D. Divsalar and M.K. Simon, "Multiple symbol differential detection of MPSK," *IEEE Trans. Communications*, vol. 38, pp. 300-308, March 1990.

[77] D. Makrakis and K. Feher, "Optimal noncoherent detection of PSK signals," *IEE Electronics Letters*, vol. 26, pp. 398-400, March 1990.

[78] M. Peleg and S. Shamai, "Iterative decoding of coded and interleaved noncoherent multiple symbol detected DPSK," *IEE Electronics Letters*, vol. 33, no. 12, pp. 1018-1020, June 1997.

[79] P. Hoeher and J. Lodge, "'Turbo DPSK': Iterative differential PSK demodulation and channel decoding," *IEEE Trans. Communications*, 47(6), pp. 837-843, June 1999.

[80] G. Colavolpe, G. Ferrari and R. Raheli, "Noncoherent iterative (turbo) detection," *IEEE Trans. Communications*, vol. 48, no. 9, pp. 1488-1498, Sept. 2000.

[81] A. Anastasopoulos and K.M. Chugg, "Adaptive iterative detection for phase tracking in turbo coded systems," *IEEE Trans. Communications*, vol. 49, no. 2, pp. 2135-2144, Dec. 2001.

[82] M. Peleg, S. Shamai and S. Galán, "Iterative decoding for coded noncoherent MPSK communications over phase-noisy AWGN channel," *IEE Proc. Communications*, vol. 147, pp. 87-95, April 2000.

[83] C. Komninakis and R. Wesel, "Joint Iterative Channel Estimation and Decoding in Flat Correlated Rayleigh Fading," *IEEE J. Sel. Areas Communications*, vol. 19, no. 9, pp. 1706-1717, Sept. 2001.

[84] B. Mielczarek and A. Svensson, "Phase offset estimation using enhanced turbo decoders," *IEEE Intl. Conf. Communications (ICC) 2002*, vol. 3, April 28-May 2, 2002, pp. 1536-1540.

[85] M.C. Valenti and B.D. Woerner, "Iterative channel estimation and decoding of pilot symbol assisted turbo codes over flat-fading channels," *IEEE J. Sel. Areas Communications*, vol. 9, pp. 1691-1706, Sept. 2001.

[86] S. Cioni, G. E. Corazza, and A. Vanelli-Coralli, "Turbo Embedded Estimation with imperfect Phase/Frequency Recovery," *Proc. IEEE Intl. Conf. Communica-*

tions (ICC) 2003, Anchorage, AK, May 2003, pp. 2385-2389.

[87] S. Cioni, G. E. Corazza, and A. Vanelli-Coralli, "Turbo Embedded Estimation for High Order Modulation," *Proc.* 3rd *Intl. Symp. Turbo Codes & Related Topics*, Brest, France, Sept. 1-5, 2003, pp. 447-450.

[88] European Space Agency DVB-S2 contribution, "*DVB-S2 Phase Jitter synthesis*," Jan. 2003.

[89] B.J. Frey, F.R. Kschischang, H.-A. Loeliger and N. Wiberg, "Factor graphs and algorithms," *Proc.* 35th *Annual Allerton Conf. Communication, Control and Computing*,", Sept. 1997, pp. 666-680.

[90] F.R. Kschischang and B.J. Frey, "Iterative decoding of compound codes by probability propagation in graphical models," *IEEE J. Sel. Areas Communications*, vol. 16, no. 2, pp. 219-230, Feb. 1998.

[91] A.P. Worthen and W.E. Stark, "Unified Design of Iterative Receivers Using Factor Graphs," *IEEE Trans. Information Theory*, vol. 47, no. 2, pp. 843-849. Feb. 2001.

[92] J. Dauwels and H.-A. Loeliger, "Joint decoding and phase estimation: An exercise in factor graphs," *Proc. IEEE Symp. Information Theory (ISIT) 2003*, Yokohama, Japan, June 29-July 4, 2003, p. 231.

[93] J. Dauwels and H.-A. Loeliger, "Phase Estimation by Message Passing," *Proc. Intl. Conf. Communications (ICC) 2004*, Paris, France, June 2004, pp. 523-527.

[94] G. Colavolpe, A. Barbieri and G. Caire, "Algorithms for Iterative Decoding in the Presence of Strong Phase Noise," *IEEE J. Sel. Areas Communications*, vol. 23, no. 9, pp. 1748-1757, Sept. 2005.

[95] L. Benvenuti, L. Giugno, V. Lottici and M. Luise, "Code-aware carrier phase noise compensation on turbo-coded spectrally-efficient high-order modulations," *Proc.* 8th *Intl. Workshop Signal Processing Space Communications*, Catania, Italy, Sept. 2003, pp. 177-184.

[96] A. Barbieri, G. Colavolpe and G. Caire, "Joint Iterative Detection and Decoding in

the Presence of Phase Noise and Frequency Offset," *Proc. IEEE Intl. Conf. Communications (ICC) 2005*, Seoul, Korea, May 16-20, 2005, pp. 720-724.

[97] W. Oh and K. Cheun, "Joint decoding and carrier phase recovery algorithm for turbo codes," *IEEE Communications Letters*, vol. 5, pp. 375-377, Sept. 2001.

[98] L. Zhang and A.G. Burr, "Iterative Carrier Phase Recovery Suited to Turbo-Coded Systems," *IEEE Trans. Wireless Communications*, vol. 3, no. 6, pp. 2267-2276, Nov. 2004.

[99] L. Zhang and A. Burr, "Application of Turbo Principle to Carrier Phase Recovery in Turbo Encoded Bit-Interleaved Coded Modulation System," *Proc.* 3$^{\text{rd}}$ *Intl. Symp. Turbo Codes & Related Topics*, Brest, France, Sept. 1-5, 2003, pp. 87-90.

[100] A. Grant and C. Schlegel, "Differential turbo space-time coding," *Proc. IEEE Information Theory Workshop (ITW) 2001*, pp. 120-122, Cairns, Sept. 2001.

[101] C. Schlegel and A. Grant, "Differential space-time codes," *IEEE Trans. Information Theory*, vol. 49, no. 9, pp. 2298-2306, Sept. 2003.

[102] V. Lottici and M. Luise, "Carrier phase recovery for turbo-coded linear modulations," *Proc. IEEE Intl. Conf. Communications*, New York, NY, USA, April 2002, pp. 1541-1545.

[103] V. Lottici and M. Luise, "Embedding carrier phase recovery into iterative decoding of turbo-coded linear modulations," *IEEE Trans. Communications*, vol. 52, no. 4, pp. 661-669, April 2004.

[104] A.P. Dempster, N.M. Laird and D.B. Rubin, "Maximum-likelihood from incomplete data via the EM algorithm," *J. Royal Statistical Soc.*, Ser. B, vol. 39, no. 1, pp. 1-38, Jan. 1977.

[105] T. Moon, "The expectation-maximization algorithm," *IEEE Signal Processing Mag.*, vol. 13, pp. 47-60, Nov. 1996.

[106] N. Noels, C. Herzet, A. Dejonghe, V. Lottici, H. Steendam, M. Moeneclaey, M. Luise and L. Vandendorpe, "Turbo Synchronization: an EM algorithm interpreta-

tion," *Proc. IEEE Intl. Conf. Communications (ICC) 2003*, vol. 4, May 11-15, 2003, pp. 2933-2937.

[107] M.J. Nissila, S. Pasupathy and A. Mammela, "An EM approach to carrier phase recovery in AWGN channel," *Proc. IEEE Intl. Conf. Communications (ICC) 2001*, vol. 7, June 2001, pp. 2199-2203.

[108] H. Nyquist, "Certain topics in telegraph transmission theory," *Trans. AIEE*, vol. 47, pp. 617-644, April 1928.

[109] C.E. Shannon, "Communication in the presence of noise," *Proc. Inst. Radio Engineers*, vol. 37, no. 1, pp. 10-21, Jan. 1949.

[110] F. Gardner, "A BPSK/QPSK Timing-Error Detector for Sampled Receivers," *IEEE Trans. Communications*, vol. 34, pp. 423-429, May 1986.

[111] H. Meyr, M. Moeneclaey and S.A. Fechtel, *Digital Communication Receivers: Synchronization, Channel Estimation, and Signal Processing*, Wiley Series in Telecommunications and Signal Processing, John Wiley & Sons, 1998.

[112] K.H. Mueller and M. Müller, "Timing recovery in digital synchronous data receivers," *IEEE Trans. Communications*, vol. 24, pp. 516-531, May 1976.

[113] J.G. Proakis and D.G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications*, 3$^{\text{rd}}$ ed., Prentice-Hall, 1995.

[114] J. Vesma, M. Renfors and J. Rinne, "Comparison of Efficient Interpolation Techniques for Symbol Timing Recovery," *Proc. IEEE Global Telecommunications Conf. (GlobeCom) '96*, London, UK, Nov. 1996, pp. 953-957.

[115] C.W. Farrow, "A Continuously Variable Digital Delay Element," *Proc. IEEE Intl. Symp. Circuits and Systems*, Espoo, Finland, June 6-9, 1988, pp. 2641-2645.

[116] C.N. Georghiades and D.L. Snyder, "The expectation-maximization algorithm for symbol unsynchronized sequence detection," *IEEE Trans. Communications*, vol. 39, pp. 54-61, Jan. 1991.

[117] R. Kötter, A.C. Singer and M. Tüchler, "Turbo Equalization," *IEEE Signal Pro-*

cessing Mag., vol. 20, pp. 67-80, Jan. 2004.

[118] J.R. Barry, A. Kavčić, S. McLaughlin, A. Nayak and W. Zeng, "Iterative Timing Recovery," *IEEE Signal Processing Magazine*, vol. 20, pp. 89-102, Jan. 2004.

[119] B. Mielczarek and A. Svensson, "Timing error recovery in turbo coded systems on AWGN channels," *IEEE Trans. Communications*, vol. 50, pp. 1584-1592, Oct. 2002.

[120] B. Mielczarek and A. Svensson, "Post-decoding timing synchronization of turbo codes on AWGN channels," *Proc. IEEE Vehicular Technology Conf.*, Tokyo, Japan, 2000, pp. 1265-1270.

[121] A.R. Nayak, J.R. Barry and S.W. McLaughlin, "Joint Timing Recovery and Turbo Equalization for Coded Partial Response Channels," *IEEE Trans. Magnetics*, vol. 38, pp. 2295-2297, Sept. 2002.

[122] A.R. Nayak, J.R. Barry and S.W. McLaughlin, "Joint Timing Recovery and Turbo Equalization for Partial Response Channels," *IEEE Int. Conf. Magnetics (Intermag) 2002*, Amsterdam, April 28-May 2, 2002.

[123] A.R. Nayak, J.R. Barry and S.W. McLaughlin, "Iterative Timing Recovery and Turbo Equalization," *Proc.* 3rd *Intl. Symp. Turbo Codes & Related Topics*, Brest, France, Sept. 1-5, 2003, pp. 19-22.

[124] J. Liu, H. Song and B.V.K.V. Kumar, "Symbol Timing Recovery for Low-SNR Partial Response Recording Channels," *Proc. Global Telecommunications Conf. (GlobeCom) 2002*, Taipei, Taiwan, ROC, Nov. 17-21, 2002, pp. 1129-1136.

[125] D.-U. Lee, E.L. Vallés, J.D. Villasenor and C.R. Jones, "Joint LDPC Decoding and Timing Recovery Using Code Constraint Feedback," *IEEE Communications Letters*, vol. 10, pp. 189-191, March 2006.

[126] X. Jin and A. Kavčić, "Cycle-slip-detector-aided iterative timing recovery," *IEEE Trans. Magnetics*, vol. 38, pp. 2292-2294, Sept. 2002.

[127] X. Jin and A. Kavčić, "Cycle-Slip Detection Using Soft-Output Information," *Proc. IEEE Intl. Conf. Communications (ICC) 2001*, Helsinki, Finland, June 11-14, 2001,

pp. 2706-2710.

[128] H. Wymeersch, H. Steendam, H. Bruneel and M. Moeneclaey, "Code-Aided Frame Synchronization and Phase Ambiguity Resolution," *IEEE Trans. Signal Processing*, vol. 54, no. 7, pp. 2747-2757, July 2006.

[129] T.M. Cassaro and C.N. Georghiades, "Frame synchronization for coded systems over AWGN channel," *IEEE Trans. Communications*, vol. 52, no. 3, pp. 484-489, March 2004.

[130] H. Wymeersch and M. Moeneclaey, "Code-aided phase and timing ambiguity resolution for AWGN channels," *IASTED Intl. Conf. Acoustics, Signal and Image Processing (SIP03)*, Honolulu, HI, Aug. 2003.

[131] S. Khan, S. Nagaraj, C. Schlegel and M. Burnashev, "Performance of a Correlation-Besed Detector for Packet Wireless Networks," in preparation for submission to *Canadian Conference on Electrical and Computer Engineering (CCECE) 2007*, Vancouver, BC, Canada, 22-26 April 2007.

[132] A.C. Johnson and N.T. Thomopoulos, "Characteristics and Tables of the Doubly-Truncated Normal Distribution", *Proc. of Production and Operations Management (POM) Society*, San Francisco, CA, April 2002.

Appendix A

Probability of Weight-2 and Weight-4 Two-Branch Error Events with Random Interleaving

A.1 Two-Branch Error Events and 8-PSK Mapping

We consider the relative probability of a two-branch error event with $d_H = 2$ compared to that with $d_H = 4$, for the serial concatenation of an outer (3,2,2) parity check code and inner differential 8-PSK modulation. A random bit interleaver without spreading is assumed. Two specific 8-PSK mappings are considered, natural mapping and an improved mapping shown in Figure A.1.

A.1.1 Error Events for Natural and Improved 8-PSK Mappings

Note that 8-PSK mappings are not regular, *i.e.,* the Hamming distance d_H is not the same between all symbols that are an equal squared Euclidean distance (SED) apart. For example, with natural mapping, symbols 000 and 001 have SED=0.586 and $d_H = 1$, while symbols 000 and 111 also have SED=0.586 and $d_H = 3$. This means that in considering error events, we cannot use the all-zeros path as a reference path but must

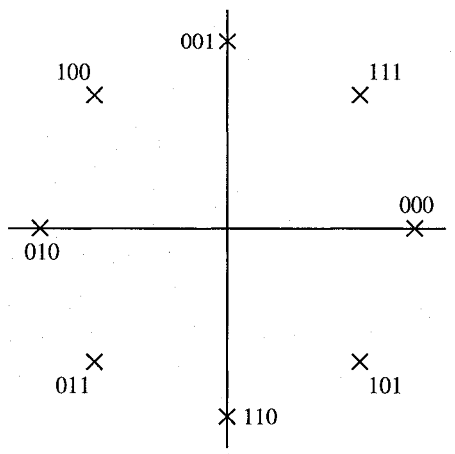


Figure A.1: An alternative 8-PSK signal mapping.

instead consider all possible paths. However, the differential 8-PSK trellis is rotationally invariant to phase rotations of multiples of $\pi/4$ rads, and thus we may consider all paths which begin in the zero state to be equivalent to their parallel rotated paths beginning in a different state.

A list of all possible correct/incorrect two-branch paths with MSED=0.586 which begin in the zero state may be determined for each mapping, and correspondingly, the Hamming distance of the error event $\mathbf{e} = \mathbf{v}' - \hat{\mathbf{v}}'$, where $\mathbf{v}'$ is the original 8-PSK mapping for the two symbols corresponding to the correct two-branch path and $\hat{\mathbf{v}}'$ is the original 8-PSK mapping for the two symbols corresponding to the incorrect two-branch path.

With natural mapping, we may consider all two-branch paths beginning in the zero state with initial correct 8-PSK symbol mapping 000 and initial incorrect 8-PSK symbol mapping 001. This consists of $\mathbf{v}', \hat{\mathbf{v}}'$= [000-000,001-111],[000-001,001-000],[000-010,001-001],[000-011,001-010], [000-100,001-011], [000-101,001-100],[000-110,000-101],[000-111,001-110]. Switching $\mathbf{v}'$ and $\hat{\mathbf{v}}'$ provides the same $\mathbf{e}$ and is equivalent. Similarly, we must consider all eight two-branch paths beginning in the zero state with initial correct 8-PSK symbol mapping 001 and initial incorrect 8-PSK symbol mapping 010, etc., and finally all eight two-branch paths with initial correct 8-PSK symbol mapping 111 and initial incorrect 8-PSK symbol mapping 000.

A.1.2 Even Hamming distance MSED two-branch error paths

In this fashion, we discover that there are 40 MSED two-branch error paths with even d_H for natural mapping, and of those, 16 have $d_H = 2$ (equivalent error events obtained by switching correct/incorrect paths are not considered). For example, [000-001,001-000],[000-011,001-010],[000-101,001-100],[000-111,001-110] all have SED=0.586 and $d_H = 2$. Note that all of these have the error path $\mathbf{e}$=[001-001]. The error paths are constrained to even d_H because of the outer (3,2,2) parity check code; each parity codeword has $d_H = 2$, and thus the entire parity-encoded sequence must have even Hamming weight. Therefore, any error path must also have even d_H.

We may do likewise for the improved mapping of Figure A.1. This mapping also has 40 MSED two-branch error paths with even d_H, but only one of these has $d_H = 2$. This is $\mathbf{v}', \hat{\mathbf{v}}'$=[010-011,011-010], with the error path $\mathbf{e}$=[001-001] - the same error pattern as for natural mapping $d_H = 2$ MSED two-branch error path.

The question may arise, if both natural and improved mappings have the same number of MSED even-weight two-branch error paths, why then do we see a lowered error floor for the new mapping if the multiplicities are the same for both mappings? The answer is that the $d_H = 2$ MSED errors are far more likely to occur in a random interleaver without spreading than are the $d_H = 4$ and $d_H = 6$ MSED errors. This will be shown in the following sections. Thus, the larger multiplicity of natural mapped $d_H = 2$ MSED errors causes a higher BER at high SNR where MSED error predominate, resulting in a higher error floor for natural mapping with respect to the new mapping.

A.2 Interleaver Design and MSED Error Events

In order to determine the relative probability of d_H=2 MSED error events in comparison with d_H=4 MSED error events, let us first examine how the structure of these error events from an interleaver perspective, including the effect of the (3,2,2) parity check code. Since the (3,2,2) parity codewords all have d_H=2, a d_H=2 MSED error event must involve only one (3,2,2) parity codeword. A possible d_H=2 error event is shown in factor graph representation in Figure A.2A, with the (3,2,2) parity-encoded bits v shown as circles, the interleaved bits v' which will be mapped to 8-PSK symbols shown as squares, and the interleaver permutations $v' = \pi(v)$ shown as edges connecting circles to squares. Figure A.2B expands each node to include the three bits of each (3,2,2) parity check codeword, shown as oval nodes, and 8-PSK symbol nodes, shown as rectangular nodes.

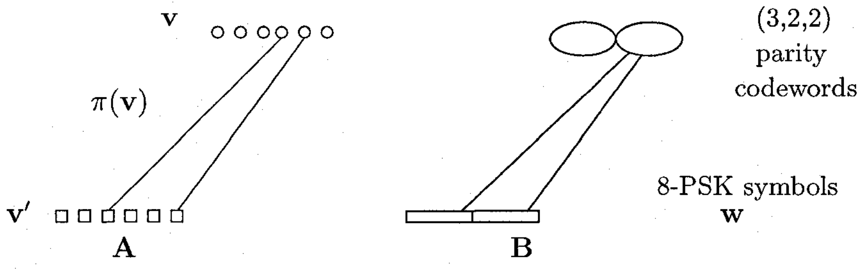


Figure A.2: Interleaver factor graph for d_H = 2 MSED two-branch events.

This factor graph representation shows clearly that the d_H=2 MSED error event involves one parity codeword and two adjacent 8-PSK symbols in a particular pattern. This error event is possible when the interleaver design permutes two bits of one parity codeword to adjacent 8-PSK symbols. An S-random spread interleaver with spreading $S \geq 3$ will obviously prevent this occurance, and provide a larger MSED=1.17, but we are considering only random interleaver design here.

A.3 Probability of a Weight-2 MSED Two-branch Error Event

The probability $P(W2)$ that a randomly designed interleaver permutes two bits of at least one (3,2,2) parity codeword to two adjacent 8-PSK symbols in the pattern _ _ x _ _ x to the last bits of two adjacent symbols, which is termed event $W2$, is calculated as follows. Assume a uniformly random interleaver of length N such that the probability any one bit of a parity codeword is interleaved to a particular bit position is $1/N$. There are $M = N/3$ codewords of the (3,2,2) parity check code and M 8-PSK symbols. A single codeword, with all 3 bit positions chosen randomly, has $\binom{N}{3}$ possible choices for bit positions.

The event $\overline{CW2}$ describes a randomly-chosen parity codeword x which does **not** have two bits permuted to two adjacent 8-PSK symbols in the pattern _ _ x _ _ x, *i.e.*, to the last bits of two adjacent symbols. The probability $P(W2) = 1 - P(\overline{W2})$, where $P(\overline{W2})$ is the probability that a random interleaver does **not** permute two bits of any parity codeword in a way that generates a weight-2 MSED two-branch error event. $P(\overline{W2})$ may be approximated by $P(\overline{CW2})^M$ for large M, as the bit permutations for different codewords become approximately independent.

Event $CW2$ describes a parity codeword that has two bits interleaved to the last bits of two adjacent 8-PSK symbols. The number of possible permutations including two consecutive last bits are found by first counting the number of possible consecutive last bit pairs, from (3,6) to (N-3,N). There are $M - 1$ consecutive last bit pairs. The third bit of the parity codeword is chosen from the $N - 2$ remaining bits. However, each set of three consecutive last bits, from (3,6,9) to (N-6,N-3,N) has been included twice. There are M-2 of these doubly-counted triples. To compensate for overcounting, they must be subtracted off once from the total. Thus the number of possible permutations with two consecutive last bits is found to be

$$(M-1)(N-2) - (M-2) = (M-1)(3M-2) - (M-2) = 3M^2 - 6M + 4. \quad \text{(A.1)}$$

The probability of event $CW2$, that one codeword permutes to two consecutive last bits of adjacent 8-PSK symbols, is found by dividing the number of possible permutations with two consecutive last bits by the total number of possible permutations; this gives

$$\begin{aligned}
P(CW2) &= \frac{3M^2-6M+4}{\binom{N}{3}} = \frac{3M^2-6M+4}{\binom{3M}{3}} \\
&= \frac{3M^2-6M+4}{3M(3M-1)(3M-2)/6} \\
&= 63M^2 - 6M + 427M^3 - 27M^2 + 6M \\
&\approx \frac{18M^2}{27M^3} = \frac{2}{3M} \quad \text{for large } M. \qquad \text{(A.2)}
\end{aligned}$$

The probability that a codeword does **not** permute to two consecutive last bits, event $\overline{CW2}$, is given by

$$P(\overline{CW2}) = 1 - \frac{2}{3M} \qquad \text{(A.3)}$$

For large M, the permutations for each parity codeword are approximately independent. Thus the probability $P(\overline{W2})$ that a random interleaver does **not** have any of its M codewords permuting to two consecutive last bits of adjacent 8-PSK symbols is found as

$$P(\overline{W2}) = P(\overline{CW2})^M = (1 - \frac{2}{3M})^M. \qquad \text{(A.4)}$$

The binomial series expansion may be used to calculate Equation A.4 for large M, as $\frac{2}{3M} \ll 1$. For good accuracy, the first five terms are included. All factors of $1/M$ or smaller are neglected.

$$\begin{aligned}
(1-x)^n &= 1 - nx + \frac{n(n-1)x^2}{2!} - \frac{n(n-1)(n-2)x^3}{3!} + \dots \\
(1-\frac{2}{3M})^M &\approx 1 - \frac{2}{3} + \frac{4(1-\frac{1}{M})}{18} - \frac{8(1-\frac{6}{M}+\frac{2}{M^2})}{162} + \frac{2(1-\frac{6}{M}+\frac{11}{M^2}-\frac{6}{M^3})}{243} \\
&\approx 1 - \frac{2}{3} + \frac{4}{18} - \frac{8}{162} + \frac{2}{243} = 0.514. \qquad \text{(A.5)}
\end{aligned}$$

Note that this result is independent of interleaver length as long as $M \gg 1$.

The probability that a random interleaver contains at least one weight-2 two-branch error event is found as $P(W2) = 1 - P(\overline{W2}) \geq 1 - 0.514 = 0.486$. Thus the probability of a randomly-generated interleaver containing at least one weight-2 two-branch error event is very high, almost 50%. This probability was confirmed by numerical simulations to be approximately 50% and is independent of interleaver length, given $M \gg 1$.

A.4 Two-branch Error Events with $d_H = 4$

We now consider MSED two-branch error events with $d_H = 4$ for both mappings. A small sampling of these error events follows. For natural mapping, some d_H=4 two-branch error events which occur with a correct bit sequence $\mathbf{v}'$ and incorrect bit sequence $\hat{\mathbf{v}}'$ are: $[\mathbf{v}', \hat{\mathbf{v}}']$=[000-000,001-111],[001-010,010-001], [011-111,100-110],[101-010,110-001]. The weight-4 error sequences $\mathbf{e} = \mathbf{v}' - \hat{\mathbf{v}}'$ are of two types: a) 001-111 or 111-001, with $d_H = 3$ on one branch and d_H=1 on the other branch; and b) 011-011, with $d_H = 2$ on both branches. Natural mapping has 20 $d_H = 4$ MSED two-branch error events, assuming a zero-state beginning; there are 15 type (a) error events, and 5 type (b) error events. All parallel sequences are equivalent. Reverses are not counted, so [000-000,001-111] is considered equivalent to [001-111,000-000]. Counting reverses, natural mapping has 40 $d_H = 4$ MSED two-branch error events.

The improved mapping of Figure A.1 has 38 possible $d_H = 4$ MSED two-branch error events, 36 with $d_H = 2$ on each branch, and 2 with $d_H = 3$ on one branch and $d_H = 1$ on the other branch. Again, parallel sequences are equivalent, and reverses are also not counted; counting reverses, there are 76 possible $d_H = 4$ MSED two-branch error events. A few of these $d_H = 4$ MSED two-branch error events are: $[\mathbf{v}', \hat{\mathbf{v}}']$= [000-110,101-101],[000-101,101-000],[011-001,110-111][110-010,101-100], [000-011,111-010].

The weight-4 error sequences $\mathbf{e}$ are of four basic types: a) 101-101, b) 101-011, 011-101, 101-110 or 110-101, c) 011-110, and d) 111-001 or 001-111, where type (d) has one branch with $d_H = 3$ and one branch with $d_H = 1$. These types are general enough that we may assume no specific patterns for the improved mapping with little loss in accuracy.

As each (3,2,2) parity check codeword has Hamming weight $d_H = 2$, a weight-4 error event must involve two parity codewords. Figure A.3 shows a factor graph representation of a weight-4 two-branch error event, where ovals represent the (3,2,2) parity codewords and rectangles represent each three bits mapping to an 8-PSK symbol. The edges connecting ovals and rectangles are the interleaver bit permutations. A weight-4 two-branch error event involves two parity codewords, with two errors each, permuting to two adjacent 8-PSK symbols. Two particular error patterns are shown in Figure A.3, with the special case of one bit error in one branch and three bit errors on the other branch shown in Figure A.3B.

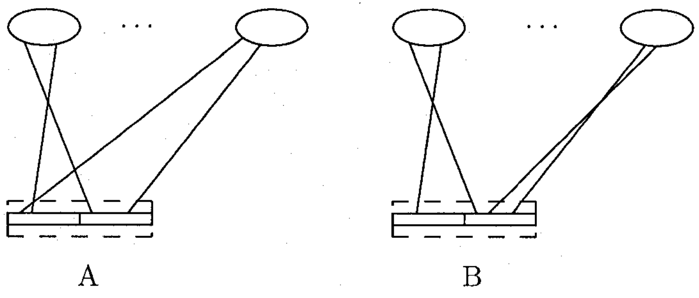


Figure A.3: Interleaver codeword/symbol factor graph for d_H=4 MSED two-branch error event.

Determining the probability that a random interleaver contains a pattern of the $d_H = 4$ two-branch error type proceeds as follows. First, we determine the probability $P(CW4)$ that between two codewords, a $d_H = 4$ two-branch error pattern exists. From that, the probability $P(\overline{CW4}) = 1 - P(CW4)$ that no d_H=4 two-branch error pattern exists is calculated. Then the probability $P(\overline{W4})$ that a random interleaver does not contain any d_H=4 two-branch error patterns is found as the probability that between all two-codeword combinations, there are no d_H=4 two-branch error patterns.

$$P(\overline{W4}) \leq P(\overline{CW4})^{\binom{M}{2}} = P(\overline{CW4})^{\frac{M(M-1)}{2}} \approx P(\overline{CW4})^{\frac{M^2}{2}}, \quad M \gg 1, \qquad \text{(A.6)}$$

where M is the number of (3,2,2) parity codewords, with M=$N/3$ and N being the interleaver length. Finally, the probability $P(W4)$ that a random interleaver contains at least one d_H=4 two-branch error pattern is found as $P(W4) = 1 - P(\overline{W4})$.

We shall assume that no specific pattern exists for the d_H=4 two-branch error event other than that two bits from each of two codewords must permute into two adjacent 8-PSK symbols. This is a good assumption for the mapping of Figure A.1; for natural mapping, it results in over-estimation of the probability of forming a d_H=4 two-branch error event, which ultimately results in over-estimation of the probability $P(W4)$ that a random interleaver contains at least one d_H=4 two-branch pattern. However, analysis will show that $P(W4)$ is significantly smaller than $P(W2)$, even using this general assumption.

Let us consider that the two (3,2,2) parity codewords x and y have between them six bits, two of which map to an 8-PSK symbol X and two of which map to either its adjacent 8-PSK symbol Y or Z. The remaining two bits may map to any remaining position. If we then view events from the permuted bits rather than the parity codeword bits, we may lump together events where one bit from both x and y go to symbols X and Y and another bit from both x and y also goes to symbols X and Y, with events where two bits from x go to X and two bits from y go to Y. We do this by considering the two permuted bits in symbol X to be of color red, and the two permuted bits in symbol Y or symbol Z to be of color blue. The two remaining bits are colored one red and one blue. We are not concerned with which codeword, x or y, the bits originally came from. Note there are $\binom{6}{3}$ ways to categorize or color the six bits of codewords x and y so that three are blue and three are red.

Symbol X may either be one of the two end symbols of the entire coded sequence, or one of the $M-2$ middle symbols. First, consider the case where symbol X is one of the two end symbols. As X is an end symbol, only one symbol position adjacent to X is available. Symbol X contains at least two red bits. Symbol Y contains at least two blue bits. We further divide this case into event A, where no red bits are in Y, and event B, where the last red bit is in Y.

For event A, there are $N-5$ positions available for the last red bit, the remaining bit position in X or any bit position in the $M-2$ remaining symbols. Similarly, the last blue bit can be in the remaining bit position in Y or any bit position in the $M-2$ remaining symbols. Thus the number of possible combinations for the red bits (also for the blue bits) is $3(N-5)-2$, with $\binom{3}{2} = 3$ different ways to place two bits in one symbol. The combination which includes all three red bits in symbol X (or all three blue bits in symbol Y) is counted three times, so to avoid overcounting, the two redundant cases are subtracted off. The total possible combinations in event A is then given by

$$A = 2(3(N-5)-2) = 2(3(3M-5)-2) = 2(9M-17) = 18M-34, \tag{A.7}$$

as there are two end symbols.

Event B, where one red bit is in Y with the other two red bits in X, has three possible choices for the single red bit in Y, and three possible combinations for the two red bits in X. There are nine total combinations for the red bits. Y contains at least two blue bits. The remaining blue bit is either in the remaining position in X or any of the $M-2$ remaining symbols, giving $N-5$ possible choices. The total combinations of event B are then found for the two end symbols by multiplying the red bit combinations by the blue bit combinations, and multiplying this term by a factor of two for the two end symbols, resulting in

$$2 \times 9(N-5) = 18(N-5) = 18(3M-5) = 54M-90. \tag{A.8}$$

If X is one of the $M-2$ middle symbols, then X has two adjacent symbols Y and Z, one to the left and one to the right. Again, symbol X contains at least two red bits; either symbol Y or Z contains at least two blue bits. The middle-symbol events may be divided as above into event C, where no red bits are in either adjacent symbol Y or Z, and event D, where one red bit is in one of the adjacent symbols Y or Z.

Event C may have the remaining red bit located in symbol X or in one of the $M-3$ non-adjacent symbols. Thus the remaining red bit has $N-8$ possible positions. There are three possible ways to position the first two red bits in X. Again, $3(N-8)$ results in overcounting the case where all three red bits are in X, so the number of possible combinations for the red bits is $3(N-8)-2$. Either adjacent symbol Y or Z may contain two or three blue bits. The blue bit scenario is the same as for event A, multiplied by two because there are now two adjacent symbols instead of one. There are $2(9M-17)$ possible combinations for the three blue bits.

Multiplying the red bit combinations by the blue bit combinations, and also multiplying by the $M-2$ possible middle symbols, we obtain for event C the following total possibilities:

$$(M-2)(3(N-8)-2)2(9M-17) = 2(M-2)(9M-26)(9M-17). \tag{A.9}$$

Event D has two red bits in symbol X and one red bit in one of the adjacent symbols Y or Z. There are 6 possible bit positions for the last red bit, and 3 possible combinations of the two red bits in X, for a total of 18 possible red bit combinations.

Two blue bits are either in the same adjacent symbol as the one red bit, or in the opposite adjacent symbol. If they are in the same adjacent symbol, then the last blue bit must be in one of the remaining $N-5$ positions. If they are in the opposite adjacent symbol, then the last blue bit can either be in the same symbol as the other two blue bits, or in one of the other symbols. This scenario is the same as for event A, resulting in $9M-17$ possible combinations. The number of bit combinations for the blue bits is thus $N-5+9M-17$ for one symbol.

The number of possible combinations for event D is found as

$$(M-2)18(N-5+9M-17) = 18(M-2)(3M-5+9M-17) = 18(M-2)(12M-22), \tag{A.10}$$

as there are $M-2$ middle symbols.

The number of total combinations in which a pair of parity codewords form a weight-4 error pattern with adjacent 8-PSK symbols is the sum of possible combinations for events

A, B, C and D, which is

$$\begin{aligned}
&= 2(9M-17)(9M-17)+54M-90+2(M-2)(9M-17)(9M-26)+ \\
&\quad 18(M-2)(12M-22) \\
&= 162M^2-612M+54M+578-90+ \\
&\quad 2(M-2)(81M^2-387M+442+108M-198) \\
&= 2(81M^2-279M+244)+2(M-2)(81M^2-279M+244) \\
&= 2(M-1)(81M^2-279M+244) \qquad \text{(A.11)}
\end{aligned}$$

The probability that a pair of parity codewords form a weight-4 error pattern with adjacent 8-PSK symbols is then found by dividing the number of total weight-4 combinations by the total possible combinations, as

$$\begin{aligned}
P(CW4) &= P(A)+P(B)+P(C)+P(D) \\
&= \frac{2(M-1)(81M^2-279M+244)}{\binom{N}{6}\binom{6}{3}} \\
&= \frac{2(M-1)(81M^2-279M+244)}{N(N-1)(N-2)(N-3)(N-4)(N-5)/36} \\
P(CW4) &= \frac{72(M-1)(81M^2-279M+244)}{N(N-1)(N-2)(N-3)(N-4)(N-5)} \qquad \text{(A.12)}
\end{aligned}$$

For large $N =$ and $M = N/3$, $P(CW4)$ is well-approximated by

$$P(CW4) \approx \frac{72(81M^3)}{N^6} = \frac{72(3N^3)}{N^6} = \frac{216}{N^3} \qquad \text{(A.13)}$$

The probability that any pair of codewords do not form a d_H=4 error pattern is found as $P(\overline{CW4}) = 1-P(CW4) = 1-216/N^3$. The probability $P(\overline{W4})$ that a random interleaver does not have any d_H=4 error patterns between any of its codewords is approximated by raising $P(\overline{CW4})$ to the power of the number of two-codeword combinations possible in the interleaver. There are $M = N/3$ parity codewords.

$$\begin{aligned} P(\overline{W4}) &\approx \left(1 - \frac{216}{N^3}\right)^{\binom{M}{2}} \\ P(\overline{W4}) &\approx \left(1 - \frac{216}{N^3}\right)^{\frac{M(M-1)}{2}}. \end{aligned} \tag{A.14}$$

The binomial series expansion is again used to approximate this term. Only two terms will be necessary because of the large powers of N; the rest may be neglected for $N > 100$.

$$\begin{aligned} P(\overline{W4}) &\approx \left(1 - \frac{216}{N^3}\right)^{\frac{M(M-1)}{2}} \\ &\approx 1 - \frac{216}{N^3}\frac{M(M-1)}{2} \\ &\approx 1 - \frac{216}{N^3}\frac{(N^2 - 3N)}{18} \\ P(\overline{W4}) &\approx 1 - \frac{12}{N}. \end{aligned} \tag{A.15}$$

The probability $P(W4)$ that a random interleaver contains at least one pattern supporting a d_H=4 two-branch error event is found as

$$\begin{aligned} P(W4) &= 1 - P(\overline{W4}) \\ P(W4) &\approx \frac{12}{N}. \end{aligned} \tag{A.16}$$

Note that $P(W4)$ is $O(N^{-1})$, contrasted with $P(W2)$ which was independent of N. For our interleaver length $N = 15000$, $P(W4) \approx 0.0008 \ll P(W2) \approx 0.486$. A random interleaver is far less likely to contain a pattern supporting a d_H=4 two-branch error event than it is a d_H=2 two-branch error event. This explains the lowered error floor of the serially concatenated code using the improved mapping of Figure 1 over natural mapping, even though both mappings have the same number of even-d_H two-branch error events. The improved mapping has far fewer (1 vs. 16) d_H=2 two-branch error events, which are far more likely to be possible in a uniformly random interleaver.

Appendix B

Moving Average Filter for Channel Phase Estimates

B.1 Exponentially-Decaying Moving Average Filter

When a time-varying phase process $\varphi(t)$ is assumed, a constant phase estimate for the entire block will be insufficient to describe the process. The instantaneous channel estimates for each symbol are time-varying but nearly uncorrelated. Correlation between a range of phase estimates can be introduced through a smoothing moving average filter. The instantaneous channel estimates $\hat{h}_k$ at sampling time kT are filtered through an exponentially decaying moving average (MA) filter $F(\omega)$ to smooth them.

For the MA filter of Equation 9.10, the filter taps f_j of the MA filter $F(w)$ are defined by $\tilde{h}_k = |F(\omega = 0)|^{-1} \sum_{j=0}^{n} f_j \hat{h}_{k-j}$ with $f_j = \alpha^j$, where $\tilde{h}_k$ is the smoothed channel estimate at sampling time kT, $\hat{h}_{k-j}$ is the instantaneous channel estimate at time $(k-j)T$, and α is the exponential decay parameter, constrained by $0 < \alpha < 1$. The value of α determines how fast the MA filter dies off. If $\alpha = 1$, all previous channel estimates are weighted equal to the current estimate, and the filter is a simple averaging filter.

The moving average filter of Equation 9.10 operates on the data in a forward direction.

This means that the first few estimates are less accurate than the last estimates as fewer samples are considered in the estimate. The moving average filter may also operate on the data in a backwards sequence, courtesy of the turbo coding environment where the entire frame of data is available to work with. The channel estimates for the backward moving average filter will be more accurate towards the beginning of the data frame, and less accurate towards the end (which are the first estimates out of the backward MA filter). Now that two sets of channel estimates are available, they may be averaged with a smoothing filter, which takes the first few channel estimates from the backward MA, the last few channel estimates from the forward MA, and smoothly ramps up to an equal average of the two MA filtered estimates. This is accomplished as

$$\tilde{h}_k = .5(\tanh((1-\alpha)k/2) * \tilde{h}_{k,\text{for}} + (1 - \tanh((1-\alpha)(N-k)/2) * \tilde{h}_{k,\text{back}}). \tag{B.1}$$

The estimates may be combined in an even simpler manner, with the forward channel estimate used for the latter half of the symbol sequence, and the backward channel estimate used for the first half of the sequence. This way, no additional filtering is required, and each estimate is used in the range where it provides the most accurate results. Simulations showed no noticeable performance loss in using this lower-complexity method of combining the forward and backward filtered channel estimates.

B.2 Filter Frequency Response

The frequency response of the MA filter $F(\omega)$ with filter taps $f_j = \alpha^j$ is given by

$$\begin{aligned} F(\omega) &= \sum_{k=0}^{N-1} f_k \exp(-jk\omega) = \sum_{k=0}^{N-1} \alpha^k \exp(-jk\omega) \\ &= 1 + \alpha \exp(-j\omega) + \alpha^2 \exp(-j2\omega) + \ldots + \alpha^{N-1} \exp(-j(N-1)\omega) \end{aligned} \tag{B.2}$$

As a geometric series, this is equivalent to

$$F(\omega) = \frac{1 - \alpha^{N-1} \exp(-j(N-1)w)}{1 - \alpha \exp(-j\ \omega)}. \tag{B.3}$$

For large N, $\alpha^{N-1} \to 0$, and the filter frequency response is well-approximated by

$$F(\omega) = \frac{1}{1 - \alpha \exp(-j\ \omega)}. \tag{B.4}$$

The DC filter response, used to normalize to unity gain, is $F(\omega = 0) = \frac{1}{1-\alpha}$. The smoothed channel phase estimate of Equation 9.10 is thus obtained, with the factor of $1 - \alpha$ in front to normalize the filter response to unity.

This exponential decay filter is a low-pass filter. The magnitude of the DC-normalized frequency response, $|F(\omega)|/|F(0)|$, is shown in Figure B.1 with $\alpha = .99$, for $0 \leq \omega/\pi \leq 2$. A magnified view of $|F(\omega)|/|F(0)|$ for $0 \leq \omega/\pi \leq 0.05$ shows that the majority of the filter energy is contained in the region $0 \leq \omega \leq \pi/50$. Figure B.1 also shows the phase response of this filter for the same α and frequency range. The filter phase is nearly linear in the range of greatest magnitude, from $0 \leq \omega \leq \pi/100$. A linear phase filter introduces a time delay, as shown by the Fourier transform relationship $\mathcal{F}(s_{n-m}) = S(\omega) \exp(-j\ m\ \omega)$, where $S(\omega) = \mathcal{F}(s_n)$. A slope m of the linear phase w.r.t. ω results in a time delay of m.

The filter delay m of this filter can be found for small values of ω (within the region of greatest energy concentration) as follows.

$$F(\omega) = 1/(1 - \alpha \exp(-j\ \omega)) = (1 - \alpha \cos(-\omega) - \alpha \sin(-\omega))^{-1} = (A \exp(j\Phi))^{-1} \tag{B.5}$$

where $A = \sqrt{1 - 2\alpha \cos(\omega) + \alpha^2}$ and $\Phi = \tan^{-1}\left(\frac{-\alpha \sin(-\omega)}{1 - \alpha \cos(-\omega)}\right) = \tan^{-1}\left(\frac{\alpha \sin(\omega)}{1 - \alpha \cos(\omega)}\right)$.

For small ω, $A \approx \sqrt{1 - 2\alpha + \alpha^2} = 1 - \alpha$ as found earlier. Using the small angle approximations $\sin(\omega) \approx \omega$ and $\tan(\omega) \approx \omega$, we obtain $\Phi \approx \alpha\omega/(1-\alpha)$. Thus, $\angle F(\omega) = -\alpha\omega/(1-\alpha)$ in the region of linear phase. The filter delay $m = \alpha/(1-\alpha)$.

Use of Equation 9.10 actually gives $\tilde{h}_{i-m}$ because of the filter delay. A delay of m

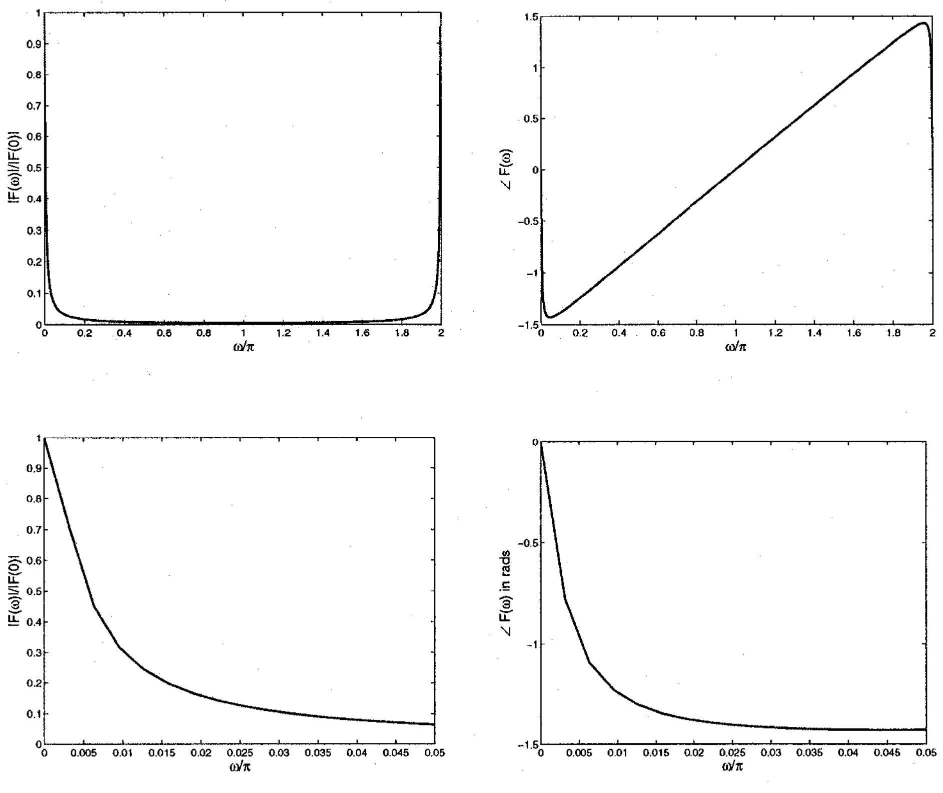


Figure B.1: Magnitude and phase response of exponentially-decaying moving average filter, $\alpha = 0.99$, for $0 \leq \omega \leq 2\pi$ and a magnified view of $0 \leq \omega \leq \pi/20$.

symbols is compensated for by using an input estimate sequence which is advanced m steps forward in time, so that $[\hat{h}_m, \ldots, \hat{h}_{k+m}]$ is used to calculate $\tilde{h}_k$ instead of $[\hat{h}_0, \ldots, \hat{h}_k]$. The filtering equation changes from Equation 9.10 to

$$\tilde{h}_k = (1-\alpha)\sum_{j=0}^{k} \alpha^{k-j}\hat{h}_{j+m}, \tag{B.6}$$

to compensate for the delay.

The last m time steps will not have a filtered channel estimate available as they would require samples that do not exist (or are in a different frame) $[\hat{h}_{N+1}, \ldots, \hat{h}_{N+m}]$ for frame length N. Two approaches may be taken. Samples from another frame may be considered not available, in which case either $\tilde{h}_{N-m}$ can be used as the channel estimate for

$[\tilde{h}_{N-m+1}, \ldots, \tilde{h}_N]$, which assumes the channel static over m symbols (not a bad assumption for small m), or a weighted combination of the channel estimate $\hat{h}_k$, $\forall i = N{-}m{+}1, N$ and $\tilde{h}_{N-m}$ may be used. Alternatively, channel samples from another frame may be available for use in the estimate. The first approach is used in the simulations presented in this thesis.