Insertion/Deletion Error Correction using Path Pruned Convolutional Codes and Extended Prefix Codes

By

Muhammad Waqar Saeed

A research report submitted to the faculty of Engineering and Build Environment
In partial fulfilment of the requirements for the degree

Masters of Science in Engineering

in

Electrical Engineering (Telecommunication)

at the

UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

Supervisor: Dr. Ling Cheng

August 2014

Declaration

I declare that this project report is my own, unaided work, except where otherwise acknowledged. It is being submitted for the partial fulfilment of the degree of Master of Science in Engineering at the University of the Witwatersrand, Johannesburg, South Africa. It has not been submitted before for any degree or examination at any other university.

Candidate Signature	: ....
Name	: ....
Date	: (Day).......... (Month).......... (Year)............

Abstract

Synchronization error correction has been under discussion since the early development of coding theory. In this research report a novel coding system based on the previous done work on path-pruned convolutional codes and extended prefix synchronization codes is presented. This new coding scheme is capable of correcting insertion, deletion and synchronization errors. A codebook has been constructed that contains synchronization patterns made up of a constraint part (maker sequence) and an unconstraint part based on the concept of extended prefix codes. One of these synchronization error patterns are padded in front of each frame. This process is done by mapping information bit to a corresponding bit sequence using a mapping table. The mapping table is constructed by using path-pruning process. An original rate convolutional code is first punctured using a desired puncturing matrix to make enough paths available at each state of the trellis. The desired paths are then pruned and matches to the extended prefix codebook constructed. The path pruning process consists of a feedback mapper attached in front of the original rate parent convolutional encoder with puncturing. The state of the convolutional encoder is fed back to the mapper which maps first information bit of the frame into a multi-bit sequence that is fed into the convolutional encoder with puncturing and thus produces one of the synchronization patterns contained within the codebook constructed. The remaining bits of the frame are encoded normally using convolutional encoding with a puncturing process only. This process is repeated periodically depending on the condition of the channel.

Simulations were performed to evaluate the ability of new system to resynchronize and correct insertion/deletion and synchronization errors at the receiver, from which favorable results were obtained. Simulations were performed with different synchronization pattern (extended prefix code word) lengths, different constraint lengths of the parent encoder and using Reed-Solomon codes as outer code in concatenation with new coding system.

A complete concatenated coding system is thus demonstrated and studied that resynchronizes and corrects insertion, deletion and substitution errors.

Dedication

I dedicate my research work to my family and friends. I owe my sincere gratitude to my loving parents, especially my father whose words of encouragement and support helped me to overcome all the difficulties and hurdles that I faced during the research.

Acknowledgement

First and above all, I praise almighty ALLAH, for providing me with this opportunity and granting me the capability to proceed successfully.

This research report appears in its current form due to the assistance and guidance of several people. I would therefore wish to thank the following people who were more than generous with their expertise and time, and helped me to complete my M.Sc. degree.

My dear parents, Prof. and Mrs. Muhammad Saeed, who have instilled a desire for formal education in me, and who have been supportive both financially and morally. I am forever indebted to you for the foundation you laid for me in life since my childhood.

Dr. Ling Cheng, my esteemed promoter, my cordial thanks for accepting me as M.Sc. student under your supervision. I am most grateful for your warm encouragement, thoughtful guidance, critical comments and for providing directions and correction of the research report. Thank you for the opportunity you gave me to develop and improve my skills and learning.

My beloved brothers Dr. Muhammad Nadeem, Col Ahmed Naeem and my brother-in-law Ishtiaq Ahmed for your encouragement and financial support.

I want to express my deep thank you to Prof. Rex Van Olst the head of Telecommunication department for his kindness, affection and financial support.

My colleagues in the School of Electrical and Information Engineering and the CeTAS research group; Muhammad Mehran Manzoor, Familua Ayokunle, Mpendulo Ndlovu, Reevana Balmahoon, Sibonginkosi Ntuli, Rohaan Shah, Abdul Wahab, Mehroze Abdullah, Benjamin Sim, Peter Brookstein, Duane Churms and those that I fail to mention, I recognize your immense contributions.

Table of Contents

List of Abbreviations

BER	Bit Error Rate
C	Channel Capacity
B	Channel Bandwidth
$\frac{S}{N}$	Signal to Noise Ratio
ARQ	Automatic Repeat Request
FEC	Forward Error Correction
EPS	Extended Prefix Synchronization
P_i	Insertion Probability
P_d	Deletion Probability
P_s	Substitution error probability
SPA	Sum-Product Algorithm
RS	Reed-Solomon code
RSC	Recursive Systematic Convolutional Code
FSM	Finite State Machine
t_{error}	Error Correcting Capability
d_{free}	Free Distance
AEM	Accumulated Error Matrix
GF	Galois Field
K	Constraint Length
r	Code rate
n	Number of Output Bits of the Encoder

k	Number of Input Bits to the Encoder
L	Extended Prefix Length
B_p	Size of the Codebook
P	Puncturing Matrix
H	Number of Paths Produced by Punctured Encoder
M	Length of Puncturing Matrix

List of Figures

List of Tables

Chapter 1: Introduction

1.0 Problem Statement

The recent developments in technology, increasing growth in machine-to-machine communication, transfer of information and the need for a better resource management has led to an increase in the demand and usage of communication systems. The growth in the number of devices has resulted in more end users sharing the limited bandwidth, and also an increase in interference which results in errors on the system~~s~~. Hence there is a need to efficiently manage system bandwidth and quality of service by preventing different error types which might consequently lead to performance degradation.

By designing efficient coding schemes a better throughput can be achieved in communication systems. The purpose of channel encoding is to efficiently use band-limited channels and provide enough information to cancel the effect of noise at the receiver.

Modern communication systems heavily rely on error correction coding. This approach started in the late 1940's with some innovative work by Shannon [1], Hamming [2] and Golay [3]. Claude Shannon presented noisy-channel coding theorem in 1948 [1]. Shannon mathematically defined the entropy of information source and capacity of a communication channel and showed that a reliable communication can be achieved over a noisy channel if the source's entropy is lower than the channel capacity.

$$C = B\ log_2\left(1 + \frac{S}{N}\right) bits/sec \qquad (1.1)$$

Where C represents the capacity of the channel, B is the bandwidth and $\frac{S}{N}$ represents the signal-to-noise ratio of the channel. The significance of equation 1.1, is that an error free transmission is possible by keeping the information rate below the channel capacity, with good error protection/correction codes.

A block diagram of the digital communication system is illustrated in Figure 1-1. Information source and the information encoder have been shown in a group which is considered as discrete information data source, similarly the source decoder and the information sink form discrete sink. The discrete channel constitutes of the modulation, demodulation and the noise source [4].

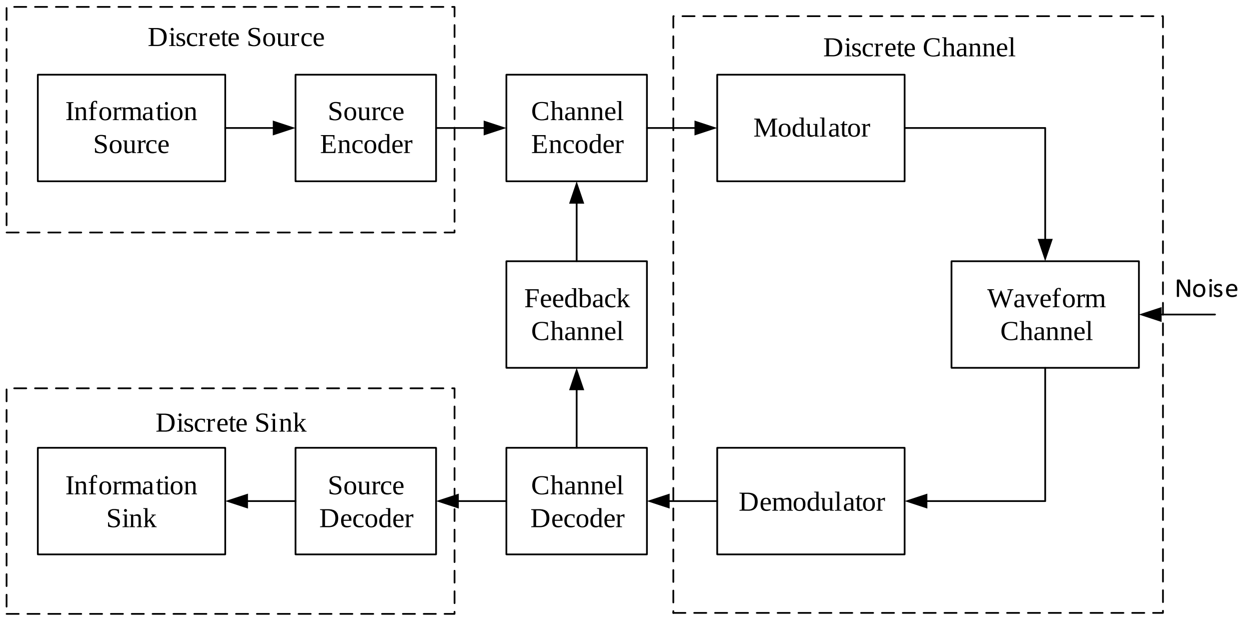


Figure 1-1: Digital Communication System [4]

All kinds of digital communication systems can be generally represented by Figure 1-1, a block diagram described in [4]. The discrete source creates bit streams that are compressed by source encoder from distinct message symbols generated by the information source. The compressed data is then coded at the transmitter by adding some redundancy such that the receiver can correct errors if introduced while passing through the channel. The most common channel encoders are convolutional and block encoders. The discrete channel constitutes of a modulator, a waveform channel and a demodulator. The modulator superimposes discrete coded symbols onto a carrier waveform, transmitted over the waveform channel at a certain frequency. The waveform channel is a continuous channel that can add noise to the carrier signal being transmitted. The most common type of noise is additive white Gaussian noise (AWGN), which causes errors in the transmitted signal. The demodulator tries to retrieve the coded information that was transmitted over the channel by superimposing on to the carrier signal. The channel decoder decodes this coded information and corrects the errors caused by the noise during transmission over the channel. The discrete sink decompresses and recovers the information that was generated by the source information.

As explained noise is a major component that affects the transmission of information over the channel. To minimize the effect of noise during transmission and to achieve reliable communication, either of the two schemes can be used: automatic repeat request (ARQ) or forward error correction (FEC). For band-limited channels ARQ can be a very costly solution because of repeated transmission of the same information. FEC is a better solution where bandwidth is very expensive and limited. FEC uses error correction coding that has the capability to correct the error at the receiver side.

These discoveries in the field of information theory and communication systems led researchers towards the error control codes. In late 1940's Golay [3] and in 1950's Hamming [2] brought in different forms of error control codes known as block codes. A decade later Bose, Ray-Chaudhuri and Hocquenghem found another class of block codes known as BCH codes [5] [6]. Peter Elias was the first to introduce Convolutional codes in 1955 [7-9]. In 1960, Reed and Solomon created the Reed-Solomon codes [10], these codes allow for excellent error correction and protection against burst errors during transmission. In 1993, Berrou and Glavieux [11] presented a practical coding scheme with a coding gain very close to that of Shannon theoretical limit.

The insertion and deletion errors can be used to represent a synchronization error channel model. The synchronization error can cause bust errors to occur unless the whole system is resynchronized. Some work has been done on insertion/deletion error correction for block codes [12-15].

Convolutional codes are very common as error correction codes and are implemented in a variety of communication systems with Viterbi decoding as the decoding algorithm. Most of the research using convolutional encoder and Viterbi decoder is bounded to substitution error correction schemes by assuming that transmitter and receiver are in synchronization. Whereas Viterbi algorithm cannot decode insertion/deletion errors correctly unless the system is synchronized.

Some work that has been done on insertion/deletion error correction using convolutional codes which can be found in [16-18].

In this research synchronization error correction using rate-compatible convolutional codes and extended prefix codes is looked into. The idea is to periodically generate extended prefix code words by puncturing and pruning an original rate convolutional encoder.

1.1 Organization of Research Report

The First chapter presents the research theme, goals and organization of the report.

Chapter 2 gives a literature survey on the work related to the research. A review is presented of work done on the construction of synchronization channel models, error control coding and synchronization codes.

Chapter 3 presents a background technical details and concepts of the system model used in this research.

In Chapter 4, techniques to design and construct a codebook of synchronization patterns and the new synchronization error correction scheme is presented. These patterns are transmitted in front of each frame. The design includes the use of marker sequence and extended prefix codes. These synchronization patterns are generated using path-pruned convolutional code. This chapter also presents the whole concatenated model of the new proposed system using path pruned convolutional codes as inner code and Reed-Solomon code as outer code.

In Chapter 5 the computer simulation and the results of the research are discussed and presented. The performance of the new system is evaluated using Bit-Error-Rate graph at various deletion probabilities. Results have been presented by varying different parameters and the conditions of the concatenated synchronization error correction system.

In Chapter 6, the research summary, conclusion and the future work of the research carried is presented.

Chapter 2: Literature Review

2.0 Introduction

A literature review in the field of forward error correction and insertion/deletion errors is presented in this chapter. First, some channel models are discussed, followed by the different types of error correction techniques that are commonly used. In the last section the recent research carried out in the field of synchronization error correction will be discussed.

2.1 Channel

A channel in telecommunication is referred to as a physical medium which includes wire, optical fiber etc. or a logical medium over air interface such as microwave, radio etc. A channel is intended to convey information from a source or transmitter to a receiver which requires some form of pathway either through a cable or a virtual broadcast media.

2.1.1 Channel Model

A channel model is the theoretical representation of a channel with certain error characterizations of a particular channel based on statistical and physical modeling. A channel can be modeled by defining its characteristics that can modify the transmitted signal passing through it. For example a wired channel can be modeled by determining effects of signal attenuation, signal interference, noise and other channel impairment on the transmitted signal. Whereas the wireless channel can be modeled by determining the effects of fading, reflection, additive noise and other channel impairment on the transmitted signal [1].

The main aim in developing a channel model is to create an artificial environment that supplies the same representation as that of a real channel. By having good knowledge of channel properties and its behavior, a more appropriate modulation and/or coding schemes can be designed that may improve error performance and achieve reliable communication by manipulating principle attributes accordingly [19].

The behavior of a system depends on the possible states of the Markov model of the system having finite or infinite states. A Markov model is a stochastic model where the state of a system changes at fixed or random interval of time and this process is probabilistic in nature. Let $S(t)$ represents the state of the system and it has n different possible values at given time such as $S_1(t), S_2(t), \dots, S_n(t)$. The current state of the system moves to next state with a certain probability. This probability is known as transition probability and it can be constant or time varying.

2.1.2 Synchronization Channel Model

Consider a *Davey-MacKay* binary channel having three parameters P_s, P_i and P_d. These parameters control the rate of substitution, insertion and deletion errors respectively [20]. A symbol can be transmitted correctly with the probability P_t, a random bit can be inserted in the sequence with probability P_i or the next bit queued in the sequence can be deleted with probability P_d. Therefore the probability of a bit transmitted is given by the equation 2.1, having a probability P_s of bearing a substitution error [21].

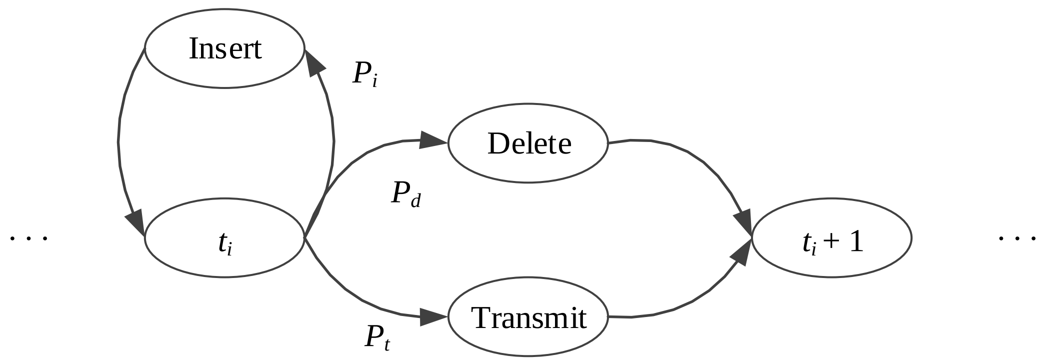


Figure 2-1: Insertion Deletion Channel with Probabilities P_i, P_d, and P_t

$$P_t = 1 - P_i - P_d \qquad (2.1)$$

Gallagar defines four scenarios that a symbol can go through while passing through a channel [22]. The symbol can be transmitted correctly with probability P_t or otherwise affected by deletion P_d, insertion P_i and substitution errors P_s given by the following equation 2.2:

$$P_t = 1 - P_i - P_d - P_s \tag{2.2}$$

Zigangirov channel model presents the channel where any number of bits can be inserted and deleted during the transmission of symbols. Substitution error is not part of the Zigangirov channel model. Hence, the probability of no bit inserted is p_i, one bit inserted is $p_i q_i$ and two bit insertion is $p_i q_i^2$ and so forth.

Therefore

$$p_i + p_i q_i + p_i q_i^2 + \cdots + p_i q_i^\infty = 1 \tag{2.3}$$

Hence,

$$p_i + q_i = 1 \tag{2.4}$$

Similarly, the probability of deletion can be given by the following equation 2.5.

$$p_j + q_j = 1 \tag{2.5}$$

Where p_j represents no deletion, $p_j q_j$ gives the probability of one insertion and so forth.

2.2 Error Control Coding

Modern communication systems heavily rely on error correction coding, this approach started in the late 1940’s with some innovative work of Shannon [1], Hamming [2] and Golay [3]. Shannon mathematically defined the entropy of information source and capacity of a communication channel and showed that a reliable communication can be achieved over a noisy channel if the source's entropy is lower than the channel capacity. The objective of error control coding is to enhance the capacity as well as reliability of a communication channel by efficiently adding carefully designed redundant data to the information being communicated over the channel known as channel coding. There are mainly two types of error control codes i.e. Block Codes and Convolutional codes.

2.2.1 Block Codes

Hamming developed the first error correction code in late 40s [23]. Hamming looked for ways to isolate and correct errors that were causing his program to halt. In the process of encoding information he grouped it into sets of four bits and added three redundant bits that act as a parity check bits. He developed an algorithm that could detect and locate the position of a single error in a block of seven encoded bits [2].

Golay addressed the problems with Hamming code and generalized its construction. He also discovered two noteworthy codes, the *binary Golay code* and *ternary Golay code* [24]. The Hamming and Golay's codes use the same scheme i.e. to group *q-ary* symbols to make a block of n symbol code word having k information symbols and $n - k$ check symbols. The error correction capability of the resultant code is t errors and code rate $r = k/n$, a code of this type is known as block code, and can be represented as a (q, n, k, t) block code. However, recent communication systems use more powerful codes instead of Golay codes. Some other linear block codes discovered were Reed-Mullar codes [25], cyclic codes [26], BCH codes [5] [27] and Reed-Solomon codes [10].

2.2.2 Convolutional Codes

The block codes discussed in the previous section have some drawbacks as well. These blocks have certain code word lengths called frames. The decoding process depends on the length of these frames, the longer the frame length the more response time required for the system to decode these blocks. Another drawback is the synchronization at the receiver end, the decoder needs to have the knowledge of the starting of each frame, i.e. which symbol is the first symbol in a received code word or frame. The third drawback is that most algebraic based decoders for block codes work with hard bit decision, rather than soft outputs of the demodulator. Hence, the performance is poor at a low signal-to-noise ratio.

The drawbacks of block codes can be avoided by using a different approach towards coding, i.e. *convolutional coding*, first introduced in 1955 by Elias [7-9]. Elias added redundancy to a continuous stream of data by using a linear shift register instead of segmenting data into blocks. In convolutional codes, each set of n output bits is a linear combination of the current set of k input bits and the m bits stored in the shift registers. The total number of bits upon which each output depends is called the constraint length. The encoder rate is the number of data bits k taken in by the encoder in one coding interval divided by the number of coded bits n during the same interval. As the data is continuously encoded, it can be continuously decoded with short response time. The encoding algorithm can make use of soft decision information as well. The first decoding algorithm was the sequential decoder of Wozencraft and Reiffen in 1961 [28], which was later modified in 1963 by Fano [29] and Jelinek in 1969 [30]. The optimal solution of maximum likelihood decoding became practical with the introduction of Viterbi algorithm in 1967 [31].

2.2.3 Concatenated Codes

Convolutional codes are susceptible to burst errors. A solution to this weakness of convolutional code is to scramble the order of the code bits by introducing an interleaver prior to transmission. This will spread the burst errors apart and will appear as independent error to the decoder. Block interleavers are most commonly used interleavers that have a $X_b \times Y_b$ bit matrix. The data is placed into the matrix column-wise and then read out row-wise or vice versa. This will make burst error length up to Y_b bits spread apart so that one error occur every X_b bits. Another type of interleaver is a cross or convolutional interleaver, which allows continuous interleaving and deinterleaving and is mostly used with convolutional codes [32].

Reed-Solomon codes handle burst errors quite well. Therefore, RS codes have properties that are complimentary to those of convolutional codes. A RS code and a convolutional code designed by concatenation in series is a very efficient system for power limited channels. Data is first encoded by an RS encoder which then goes in to the convolutional encoder. At the receiver end the convolutional decoding is performed first and then fed into the RS decoder. Therefore, each decoder performs its suitable operation on the data i.e. convolutional decoder works with independent errors with low SNR, while RS decoder works with burst errors and high SNR. David Forney in 1966 proposed this method of serial concatenation. [33].

It is found that serial concatenated codes offer comparable performance and in some cases better to that of parallel concatenated codes [34]. The performance i^{th} convolutional component codes can also be matched or exceeded with block component codes such as Reed-Solomon [35], Hamming [2] [36] and BCH [5] [37] codes.

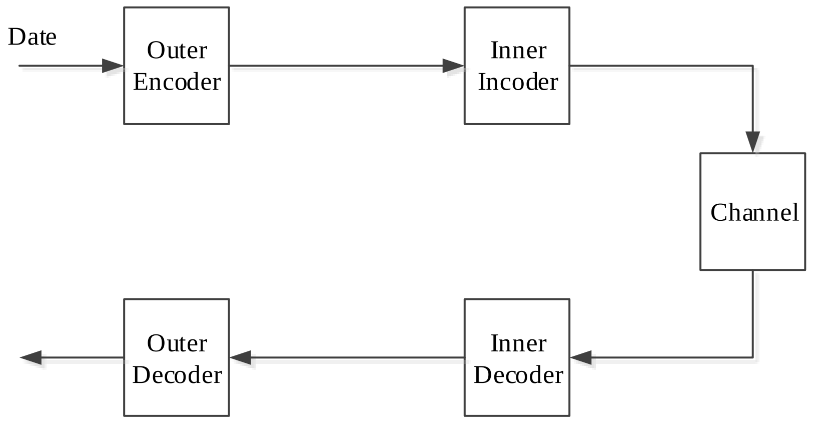


Figure 2-2: A serial Concatenated Code

2.2.4 Turbo Codes

Berruou, Glavieux and Thitimajshima in June 1993 presented a new coding scheme at International Conference on Communication in Geneva Switzerland. This new coding scheme was able to achieve a practical code rate very close to that of Shannon's theoretical limit. They presented the new class of codes and its decoding technique named “Turbo Codes” [11]. This coding technique constitutes two or more component codes combined in parallel and are from a subclass of convolutional codes known as recursive systematic convolutional (RSC) codes [11]. A turbo encoder and decoder is shown in Figure 2-3 and Figure 2-4 respectively [38].

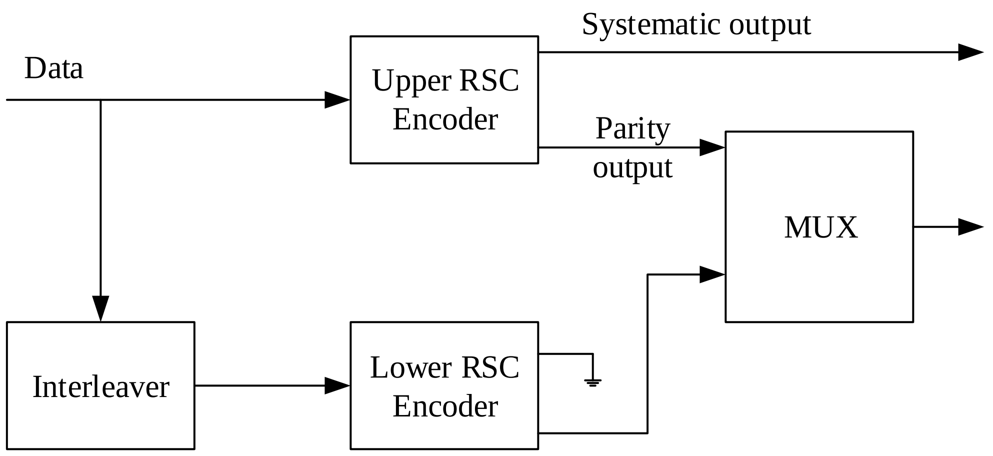


Figure 2-3: A Turbo Encoder [38]

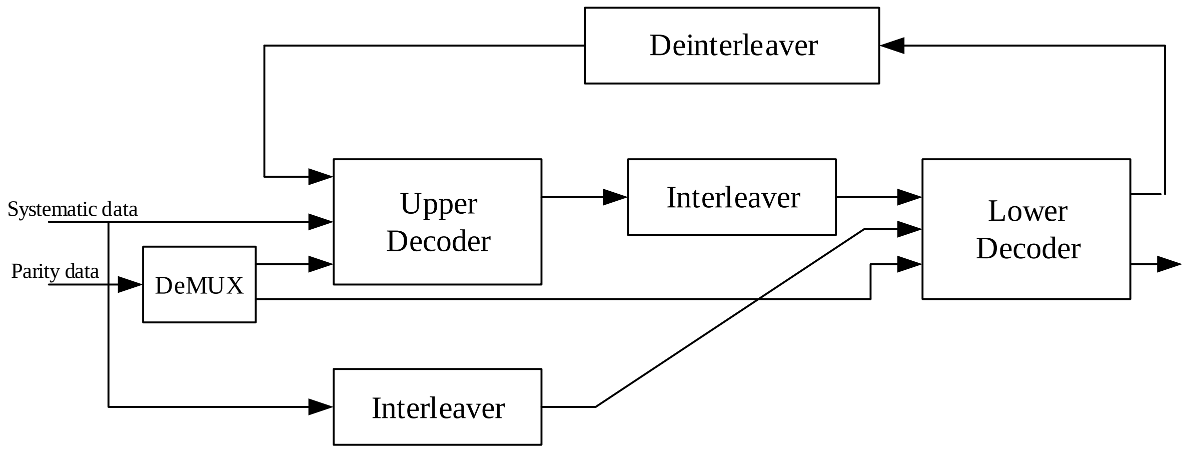


Figure 2-4: A Turbo Decoder [38]

Figure 2-4 above shows that the input is interleaved before it is fed into the lower encoder. The output of the lower encoder is redundant because both encoders receive the same input but in different order, therefore they are systematic encoders. The output of the lower encoder does not need to be transmitted. The code rate of the whole system is $r = 1/3$. Higher rates can be achieved by puncturing.

A suboptimal iterative decoding algorithm was presented in [11]. This algorithm operates at a much lower complexity, as the presence of the interleaver makes optimal decoding (maximal likelihood) of turbo codes complex and impractical. The idea behind the suboptimal decoding algorithm is to break it down into two smaller codes. The decoding of these codes is performed locally and the information is shared in an iterative fashion.

2.3 Synchronization Error Correction Codes

In this section the discussion will be on some existing techniques on insertion/deletion error correction codes.

Synchronization error correction schemes have gained more attention due to the applications such as image watermarking [39] and bit-patterned magnetic media [40]. In 1966 Levenshtein described, a code that is capable of correcting *x* number of deletions should also be able to correct *x* number of insertions and/or deletions [41]. When a message is carried as blocks of binary symbols, the need is to provide some means for the receiver to detect the beginning or the end of each block to keep synchronization with the transmitter. Generally, special synchronization symbols are used that represent a third kind of information neither 1 nor 0. The examples of such type of symbols are Morse code spaces and teletype beginning and end pulses in which each symbol is represented by a unique combination of sequences [42].

2.3.1 Prefix Codes

A receiver turned on in the middle of a transmitted message can decode data wrong if synchronization scheme is not used. Gilbert first introduced the synchronization of binary messages in 1960 known as prefix codes [15]. According to Gilbert, a short sequence known as prefix P can be used with each code word transmitted to determine the boundary of a code word. The constraint in using a prefix code is that the sequence P cannot appear in the remaining part of the block. The prefix should be chosen such that it satisfies the constraints of different blocks of

N bits (consisting of prefix P and the unconstraint part) used. Longer prefix will affect the length of message bits as N is a fixed value and corresponds to an optimum length of the prefix.

Sellers [14] introduced marker codes in 1962, which was a first major achievement in the field of insertion, deletion and substitution error correction coding. A sequence of bits called Marker codes is inserted periodically during the transmission of code words to help the receiver determine the synchronization.

Van Wijngaarden, and Morita, presented a new type of synchronization code, known as extended prefix synchronization code (EPS) in [43]. The EPS is constructed by using a so called extended prefix with fixed symbol positions and unconstraint data information positions followed by constraint data sequence. EPS extends the set of available prefix for frame recognition and data mapping procedure rather than having a single prefix used in normal prefix synchronization codes.

In *Extended Prefix Synchronization EPS-code* there are $h - k$ unconstraint positions of data, and an extended marker P (length h) having k fixed positions is used. According to Guibas and Odlyzko [44], when using q-ary PS-codes, the set of q^{h-k} different prefixes is given by P. These PS-codes can be presented as $Cp\ (k + m) = P\ Fp(m)$, where $Fp(m)$ is the set of constrained code word c_i, (where $i\ =\ 1 \dots\ m$) and P must not appear as a part of constrained code word.

2.3.2 Convolutional Codes

The convolutional codes are mainly used for the substitution error correction. Little research has been done on convolutional codes as insertion/deletion error correction code. In this section research done on this topic in the last decade will be discussed.

In [45] Swart and Ferreira proposed a new insertion/deletion error correction scheme that was based on a parallel convolutional encoder. Cheng and Ferreira presented rate-compatible convolutional codes with the Levenshtein distance metric for insertion, deletion and substitution errors [46]. The Levenshtein distance metric is asserted to be suited to use with Viterbi decoding as a branch compare metric. A new type of Viterbi decoding algorithm was presented that uses

Levenshtein distance metric for decoding and rate-compatible pruned codes for encoding. The detail of the algorithm and its applications on different channels can be found in [18] [46] and [47]. Cheng, Ferreira and Swart [48] presented a bidirectional Viterbi decoding algorithm that uses the Levenshtein distance metric and is used with a regular convolutional codes. This system has a capability of correcting an average of 30 deletions with in a 6000 bits long frame when used with $r = 0.67$ rate convolutional code [48].

A post-modulation scheme to correct insertion deletion substitution errors was presented by Cheng and Ferreira [49]. They used run-length-limited Levenshtein codes for a dc^2 – balanced code in conjunction with interleaving techniques. This protects code words from insertion, deletion and substitution errors.

2.3.3 Linear and Cyclic Codes

Linear and cyclic codes are mainly used to correct substitution errors. Insertion and deletion errors can reduce the quality of service (QoS) of the system to a very bad extent. Abdel-Ghaffar, Ferreira and Cheng in [50] and [51] investigated linear and cyclic codes for synchronization errors correction. They showed that linear code of rate greater than $1/2$ cannot correct insertion and deletion errors. They also showed that by adding an extra symbol to the code word of a cyclic codes of rates $1/3$ or $1/2$ have the potential to correct a single deletion or insertion [50]. In another approach they presented a cyclic code of rate at most $1/2$ which was shortened by deleting code words such that the shortened code was capable of correcting insertion and deletion errors. [51].

2.3.4 Number Theoretic Codes

Ferreira, Abdel-Ghaffar, Cheng *et al* presented their work on systematic encoding of number theoretic codes to develop moment balancing templates by extending a block or convolutional code with predetermined error correction capability [52] [53]. Insertion/deletion correction can be achieved by adding redundant bits at selected positions to balance the moment of the code word by using some number theoretic constructions. They investigated bit error rate performance comparison of LDPC and Convolutional codes based on sum-product algorithm (SPA) decoding and 3-bit quantization Viterbi decoding respectively.

2.3.5 Synchronization using Permutation Codes

Slepian introduced variant I and variant II permutation codes for reliable communication over certain class of noisy channels [54], Dunn used variant I codes for memory less Gaussian sources [55]. Later Berger *et al* developed permutation codes for more general sources [56-58].

A fast synchronization coding scheme was presented in [59], they have used single insertion/deletion error correcting permutation codes. The author also presented a new algorithm for permutation coded sequences which combines the dynamic algorithm and a Viterbi like decoding algorithm [60].

2.4 Summary

A brief literature survey covering synchronization channel model, general error control coding and synchronization error correction codes was given in this chapter. These coding schemes include block codes, convolutional codes, concatenated codes, turbo codes, prefix codes, number theoretic codes, and permutation codes.

Chapter 3: Background

3.0 Introduction

The research work presented in this research report is novel and based on the work previously done by Cheng in [61]. Previous work done has been presented to create a background for the new work done in this research. This chapter presents a detailed background of the channel model and encoding techniques used in this research and previously presented in [61]. The role of this research will be defined by explaining the error model used for the insertion, deletion and substitution error protection, and the coding scheme. A block diagram determining the components of the system used in this research is shown in Figure 3-1.

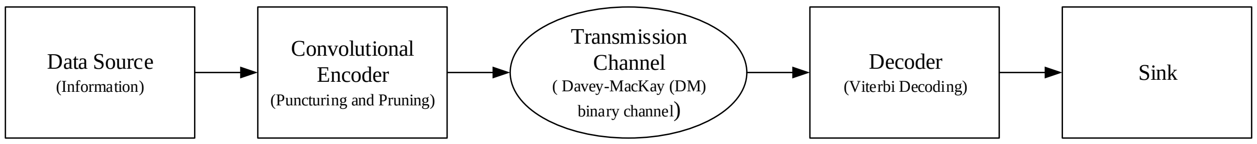


Figure 3-1: Block Diagram of the System

3.1 Prelude Definitions

Definition 1: An insertion error is the insertion of a bit(s) in the sequence resulting in the addition of an extra bit while transmitting the symbol over the channel.

Definition 2: A deletion error is when a bit is deleted from the sequence, hence resulting in a shortened or an empty word in the sequence while transmitting the symbol over the channel.

Definition 3: A substitution error is the replacement of a bit with another while transmitting the symbol over the channel.

3.2 Channel Model

A Davey-Mackay (DM) binary channel [12] was used in this research. The binary channel can be defined by three error parameters P_t, P_i and P_d which refers to the transmission, insertion and deletion probabilities. Therefore, the probability of a bit transmitted is given by the equation 3.1 shown as follows, and it has a probability P_s of bearing a substitution error [19].

$$P_t = 1 - P_i - P_d \tag{3.1}$$

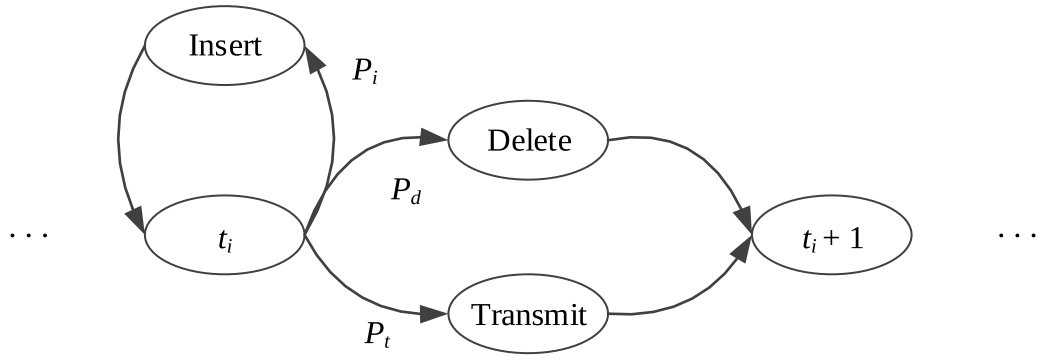


Figure 3-2: Insertion Deletion Channel with Probabilities P_i, P_d, and P_t

3.3 Convolutional Codes

A brief introduction on the error correction technique used in this research is discussed. The main focus of this research is to use rate-compatible convolutional codes to overcome the synchronization error that occurs during transmission.

There are two common types of error correcting codes available, namely: block codes and convolutional codes. A binary convolutional code is denoted as a three-tuple (n, k, m), where n is the output bits, k is the input bits and m represents the memory of the convolutional code. A detailed introduction of convolutional codes can be found in [62].

3.3.1 Convolutional Encoder

Convolutional encoder consists of shift registers that are serially connected to form a finite state machine (FSM) that processes information bits serially. Therefore, the output of the encoder depends on the input and the current state of the encoder. Each message bit influences a span of $n(m+1)$ successive output bits known as *output constraint length*. For an $(2,1,3)$ encoder, 8 successive output bits are influenced by a single input. Figure 3-3 represents a $(2,1,3)$ convolutional encoder having 3 shift registers.

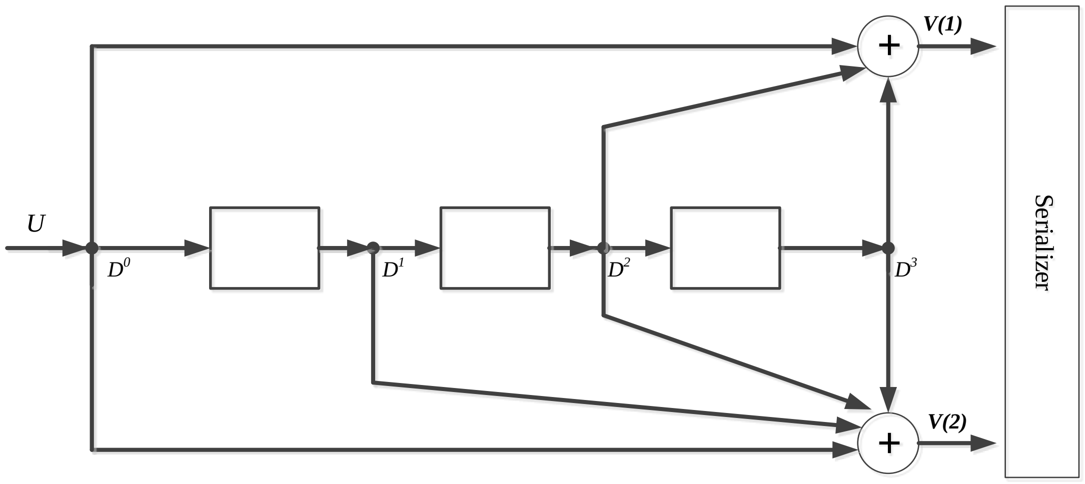


Figure 3-3: A $(\mathbf{2},\mathbf{1},\mathbf{3})$ Convolutional Encoder

3.3.2 Generator Matrix

Convolutional code can be defined by the generator sequences $g^{(1)}, g^{(2)}, \dots g^{(n)}$ that represents the output of the encoder on each input. The code word is produced by the matrix multiplication of input data and the generator matrix which is associated to the generator sequence.

The convolutional code can be generated by multiplying the information sequence by the generator matrix. Let $u_1, u_2, u_3, \dots, u_k$ be the information sequence and let $v_1, v_2, v_3, \dots, v_n$ be the output sequence.

Then

$$u = (u_{1,0}, u_{2,0}, \dots, u_{k,0}, u_{1,1}, u_{2,1}, \dots, u_{k,1}, \dots, u_{1,i}, u_{2,i}, \dots, u_{k,i}, \dots) \tag{3.2}$$

$$v = (v_{1,0}, v_{2,1}, \dots, v_{n,0}, v_{1,1}, v_{2,1}, \dots, v_{n,1}, \dots, v_{1,i}, v_{2,i}, \dots, v_{n,i}, \dots) \tag{3.3}$$

The relationship between input and output can be described as:

$$v = uG \tag{3.4}$$

Where *G* is the generator matrix of the code.

The generator matrix is given as:

$$G = \begin{bmatrix} G_0 & G_1 & G_2 & \dots & G_m & & \\ & G_0 & G_1 & \dots & G_{m-1} & G_m & \\ & & G_0 & \dots & G_{m-2} & G_{m-1} & G_m \\ & & & \ddots & & & \ddots \end{bmatrix}$$

Generator Polynomial:

The generator sequence can also be represented as polynomials. For the encoder shown in Figure 3-3 the polynomial is:

$$\begin{cases} g^{(1)}(D) = [1 + D^2 + D^3] \\ g^{(2)}(D) = [1 + D + D^2 + D^3] \end{cases} \tag{3.5}$$

An (n, k, m) encoder can be represented by a $k \times n$ matrix $G(D)$, known as polynomial generator matrix in which each entry is a polynomial.

$$G(D) = \begin{pmatrix} g_1^{(1)}(D) & g_1^{(2)}(D) & \dots & g_1^{(n)}(D) \\ g_2^{(1)}(D) & g_2^{(2)}(D) & \dots & g_2^{(n)}(D) \\ \vdots & \vdots & \vdots & \vdots \\ g_k^{(1)}(D) & g_k^{(2)}(D) & \dots & g_k^{(n)}(D) \end{pmatrix}$$

Definition: The code rate is defined as $r = k/n$, where *k* is the number of input bits and *n* is the number of outputs.

Definition: The *Hamming distance* of two sequences with the same length is the number of positions at which these two sequences differ.

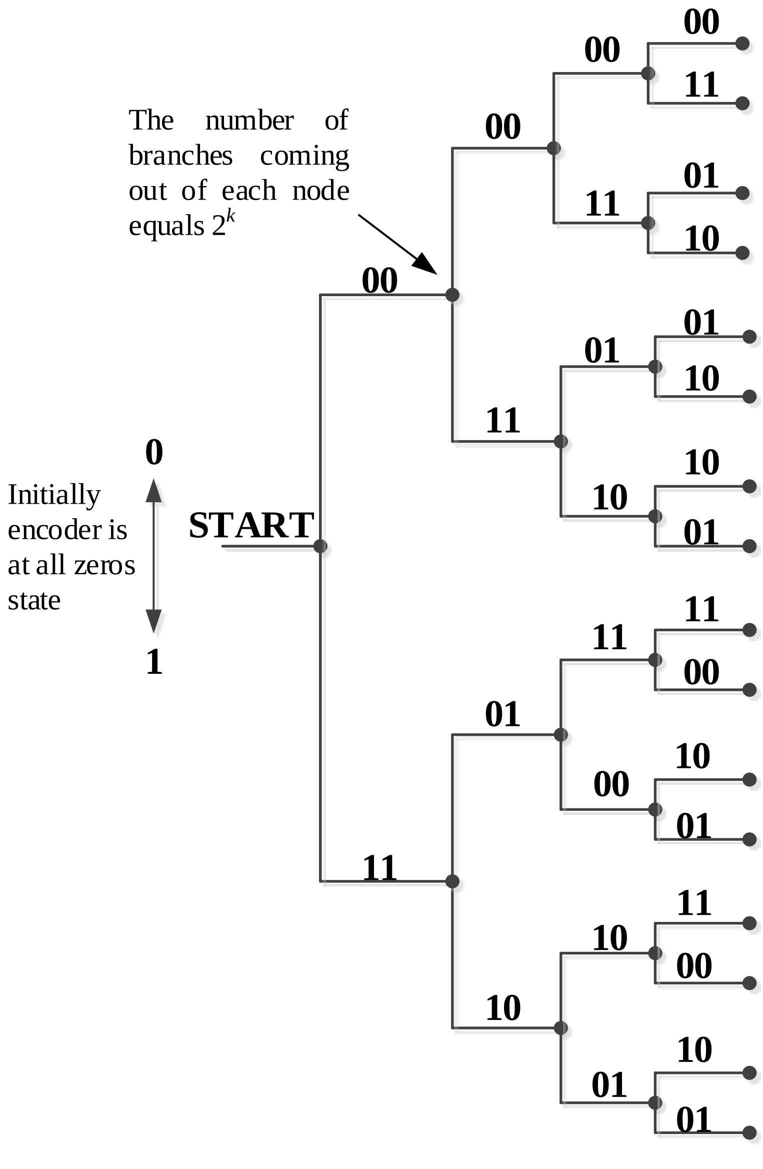
Definition: The *error correcting capability* (t_{error}) of a convolutional code is the number of errors that can be corrected by the code. It is given as

$$t_{error} = \left\lfloor \frac{d_{free} - 1}{2} \right\rfloor \tag{3.6}$$

The definition of *free distance* d_{free} will be addressed in section 3.4.1.

3.3.3 Graphical Representation of Convolutional Code

A convolutional code can be represented as a code tree. The tree generated by the encoder in Figure 3-3 is shown in Figure 3-4.



The number of branches coming out of each node equals 2^k
Initially encoder is at all zeros state
0
1
START
00
11
00
11
01
10
00
11
01
10
11
00
10
01
00
11
01
10
01
10
10
01
11
00
10
01
11
00
10
01

Figure 3-4: Tree structure of convolutional code

Figure 3-4 describes the structure of the tree diagram. As we discussed in the previous section, the convolutional encoder starts with all registers at zero, therefore the tree diagram also starts at all zero state. Each branch of the tree transforms a single bit input into a two bit output. The upper branch at each node represents 0_2 input and the lower branch 1_2 input.

Figure 3-5 describes the structure of a convolutional encoder in the form of a state diagram. The solid line represents 0_2 input and the dotted lines represents 1_2 input. Similarly Figure 3-6 represents the trellis structure of the convolutional encoder presented in section 3.3.1.

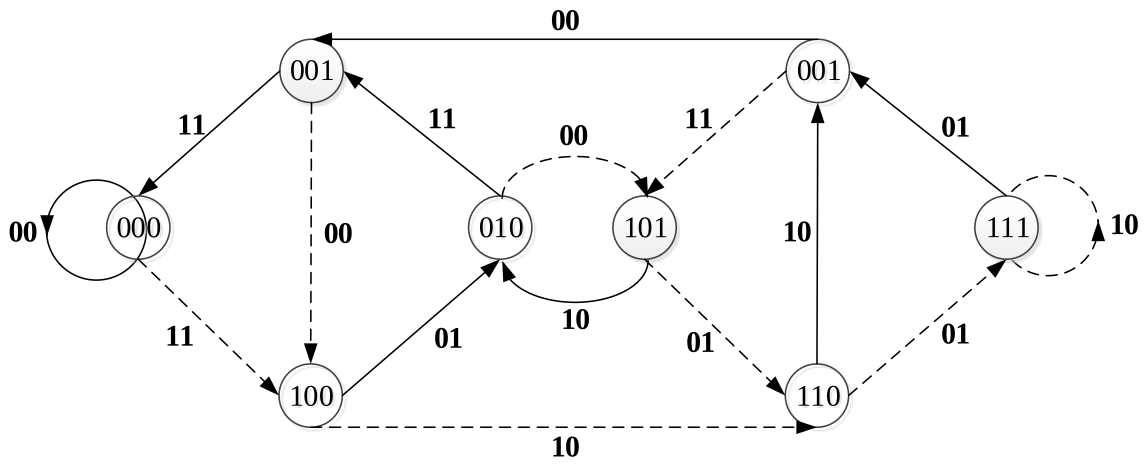


Figure 3-5: State diagram of the convolutional code

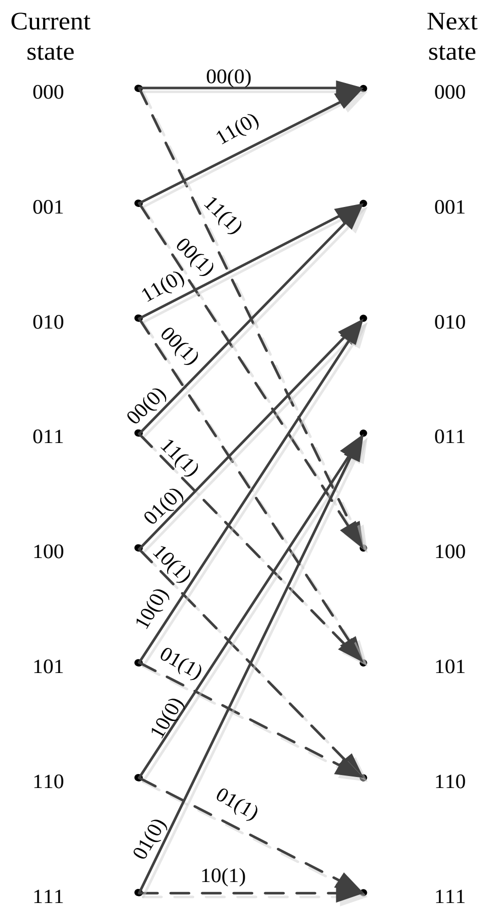


Figure 3-6: Trellis diagram of a convolutional code

3.4 Rate-Compatible Convolutional Codes

The purpose of designing a coding scheme is to achieve maximum throughput and error correction capability for a worst case scenario of the channel. In some harsh channel cases, the channel status fluctuates drastically in time and can cause a constant rate coding scheme to fail. This prompts a variable rate coding scheme that can adjust to the channel conditions to have a changing error correction capabilities. The advantage of using rate-compatible convolutional code is that the same single encoding and decoding system can be used for a range of code rates. The rate-compatible convolutional codes are obtained by using puncturing or pruning operations [18] [63].

3.4.1 Puncture Convolutional Codes

The process of puncturing is done by periodically deleting (or puncturing) encoded symbols from ordinary convolutional encoded sequence of data. Therefore, the rate of the encoder increases by puncturing process. Consider an encoder with generator polynomial:

$$G = (1 + D^2 + D^3 \quad 1 + D + D^2 + D^3) \tag{3.7}$$

The puncturing matrix $P = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$ indicates that the first bit of second encoded block is punctured.

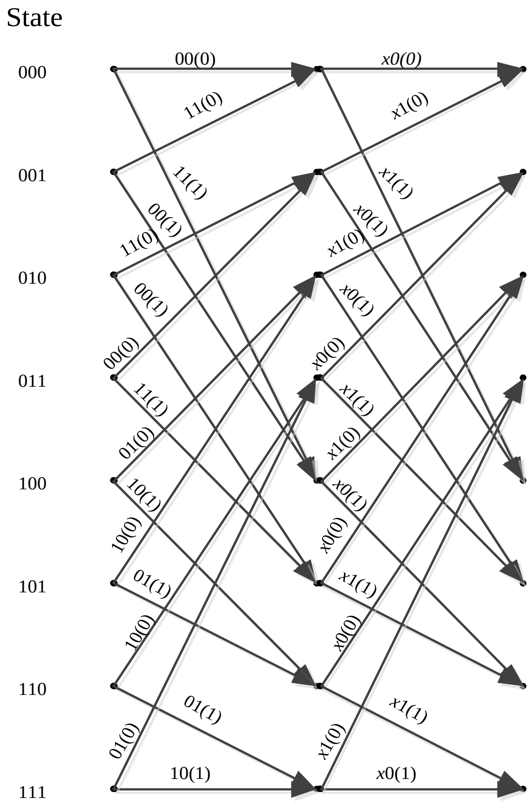


Figure 3-7: Trellis diagram of a punctured convolutional code

The puncturing process is shown in Figure 3-7. The first bit of the second trellis is the punctured/deleted bit denoted by x. The $r = 1/2$ encoder is punctured into an $r = 2/3$ encoder. The equivalent $r = 2/3$ encoder trellis diagram is shown in Figure 3-8.

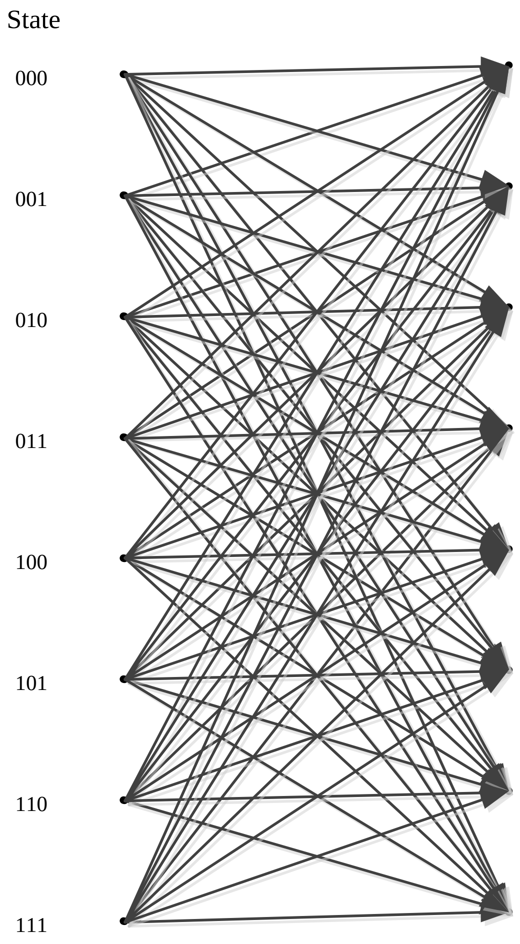


Figure 3-8: Trellis diagram of a $r = 2/3$ convolutional encoder

Puncturing reduces the free distance of the code but it is still comparable with the ordinary convolutional code having the same rate as after puncturing.

3.4.2 Pruned Convolutional Codes

Puncturing the number of branches connecting to a state increases the rate of a code. We can remove some of the undesired branches of the trellis. This process is known as pruning. Pruning is done to improve the error correcting capability or correcting special types of errors, such as insertion and deletion errors [18] [64]. Due to the trellis structure of convolutional codes, it is easy to delete some or all of the undesired code words or associated paths from it. It is considered as an inverse operation of the puncturing, thus by doing so the code rate is reduced.

Figure 3-9 describes the pruning process which consists of a feedback mapper. The state of the convolutional encoder is fed back to the mapper and the mapper maps the information data into a corresponding input by using a mapping table. This input is then encoded by the convolutional encoder that produces a corresponding code word which is part of the pruned paths.

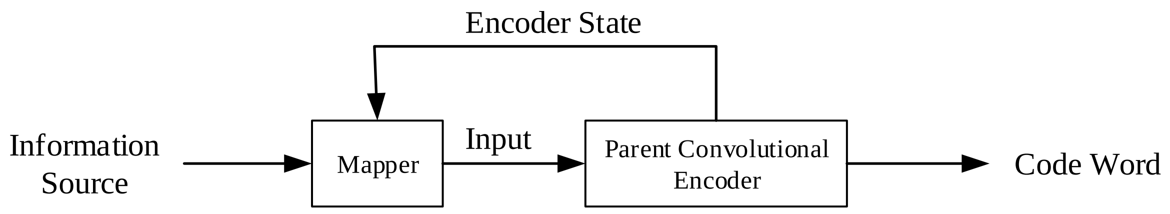


Figure 3-9: Pruning Processing Using Feedback Mapping

3.5 Decoding

As discussed above, rate-compatible convolutional codes has been used as the encoding technique in this research. Maximum-likelihood algorithm is a very widely used decoding process for convolutional codes, also known as Viterbi decoding algorithm. In this research, Viterbi decoding is used as the decoding technique for the convolutionally encoded information at the receiver.

3.5.1 Viterbi Decoding Algorithm

Viterbi decoding algorithm is a scheme for decoding convolutional codes on the basis of maximum likelihood decoding [65]. The encoded information signal is corrupted by noise when sent via a channel; the receiver tries to recover the sent sequence into the most likely sequence. This process is known as maximum likelihood decoding. The Viterbi decoding algorithm takes the two code words i.e. the received code word and the trellis branch code word, compares them for the hamming distance and selects the branch with the minimum Hamming distance [18]. An example of Viterbi decoding algorithm follows:

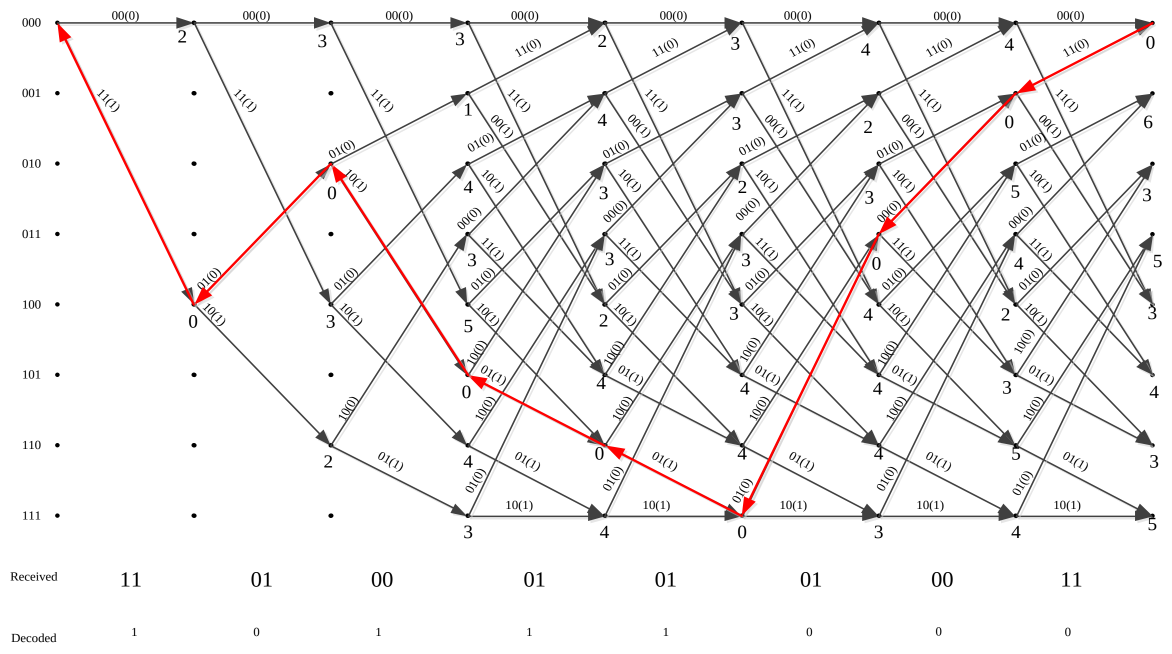


Figure 3-10: Example of hard-decision Viterbi decoding

Definition: The *Free distance* d_{free} of a convolutional code is the minimum weight of a path that starts at all zero state and terminates at all zero state.

3.6 Summary

A detailed background on synchronization channel model and convolutional codes was given in this chapter. Further, the rate compatible convolutional codes and Viterbi decoding process used in this research were discussed.

Chapter 4: Codebook Design

4.0 Introduction

In this chapter the design and construction of the codebook for the proposed system used in the research is discussed.

Consider a concatenated coding system with an inner code having capability of correcting synchronization and substitution errors and a non-binary outer code that can correct remaining substitution errors. The inner code is designed using rate-compatible convolutional codes. The scheme consists of a set of extended prefix sequences generated using path pruned convolutional codes. These extended prefix sequences are transmitted at the start of every frame. The decoder recognizes these combinations and keeps in synchronization with the encoder.

4.1 Codebook Design

A novel coding scheme has been proposed in this research. This coding scheme is designed using path pruned convolutional codes and extended prefix codes, capable of correcting synchronization errors (insertion, deletion and substitution).

Consider a $r = 1/2$ rate parent convolutional encoder with constraint length $K = 3$ punctured code using the puncturing matrix P. The length of the puncturing matrix depends on the selection of the marker sequence and the constraint length of the parent encoder. The modified code rate after puncturing will be $k/(k+1)$, where k is the number of input bits and $(k+1) = n$ is the number of output bits of the punctured convolutional code. Hence, the length of the extended prefix code word that will be selected should be n bits long. The constraint part of the extended prefix consists of a marker sequence. The unconstraint part of the extended prefix depends on the constraint length K of the parent encoder. Hence, the length of the constraint part i.e. the marker sequence is $n - K$.

The size of the codebook B_p generated from PS-code is given by:

$$|B_p| = 2^k \tag{4.1}$$

Example 1

Let us take an example of a codebook that was used for computer simulations.

The first step is to define the constraint length and the rate of the original code being used. In this example the constraint length of $K = 3$ and the code rate of $r = 1/2$ parent convolutional encoder is used.

The second step is to define the length M of the puncturing matrix P. For this example assume that $M = 7$, a different length of M can also be assumed depending on the condition of the channel. As overall code rate of the proposed coding scheme is $k/(k+1)$, the extended prefix length L (code words each having length $n = k + 1$) will therefore be $n = M + 1$ which is 8. Hence, the constraint part of each code word in the codebook will be $n – K = 5$.

As the constraint and unconstraint portion of the each extended prefix code word is known, the next step involves the construction or selection of a suitable marker sequence (constraint part of the extended prefix code word). The marker sequences are usually constructed by combining consecutive 1s and 0s. Some example marker sequences are given in the table below:

Marker Sequences
110
11100
1111000
111110000
1111110000

Table 4-1: Example Marker Sequences

After selecting a suitable marker sequence, the next step is to combine this marker with the unconstraint part to form a codebook. Table 4-3 shows the codebook constructed in this example. The size of the codebook B_p constructed is given by:

$$|B_p| = 2^3$$

Marker Sequence (Constraint Part)	Unconstraint Part	Extended Prefix Code Word
11100	000	11100000
11100	001	11100001
11100	010	11100010
11100	011	11100011
11100	100	11100100
11100	101	11100101
11100	110	11100110
11100	111	11100111

Table 4-2: Codebook

4.2 Codebook Search Methodology

After designing the codebook the next step involves searching these paths (code words) and its corresponding inputs to the encoder.

Consider the transition table shown in Table 4-3 for rate $r = 1/2$ convolutional encoder with constraint length $K = 3$, given below:

Initial State	**Input**	**Next state**	**Output**
00	0	00	00
00	1	10	11
01	0	00	11
01	1	10	00
10	0	01	01
10	1	11	10
11	0	01	10
11	1	11	01

Table 4-3: Transition Table for $r = 1/2$ and $K = 3$ Convolutional encoder

As mentioned in the previous example the length of the puncturing matrix is denoted by M. Since the length of each code word in the codebook depends on the length of the constraint marker sequence, and for the unconstraint part on the constraint length of the parent convolutional encoder, M therefore is equal to the length k which is the number of input bits to the encoder.

i.e. $M = k$

Figure 4-1 below is the trellis representation of a k/n punctured convolutional encoder. T_i represents the i^{th} puncturing interval, where $i = 1, 2, \dots, M$.

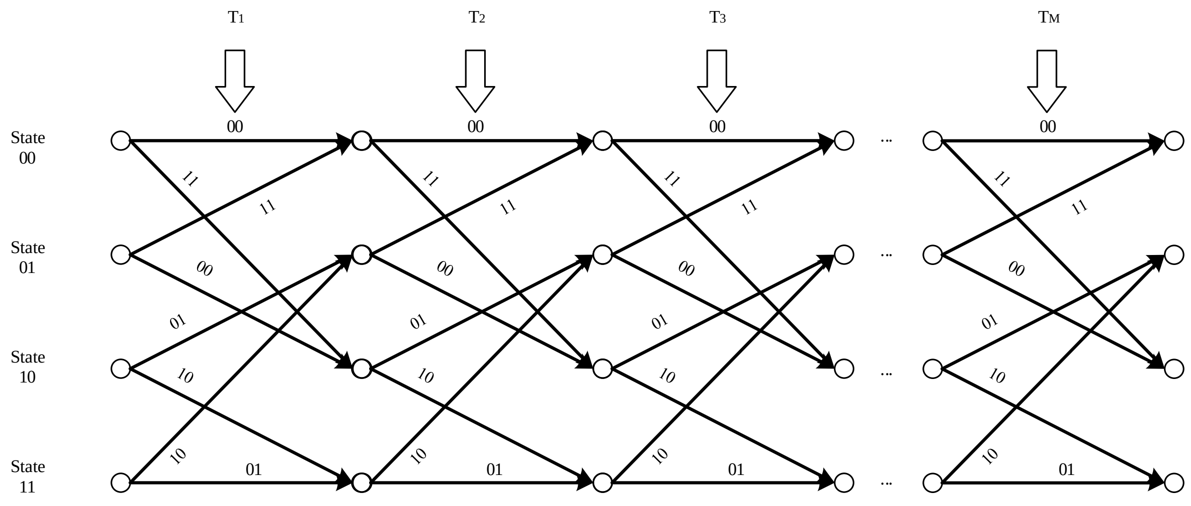


Figure 4-1: Trellis diagram with puncturing length $M, r = k/n$ and $K = 3$

The above punctured encoder will produce 2^k different paths (code words each having length $n = k + 1$) from every input state of the encoder. The number of paths H produced by the above punctured encoder can be found by:

$$H = 2^k \times 2^{(K-1)} \qquad (4.2)$$

Where K is the constraint length of the parent convolutional encoder and k, is the number of input bits of the punctured convolutional encoder.

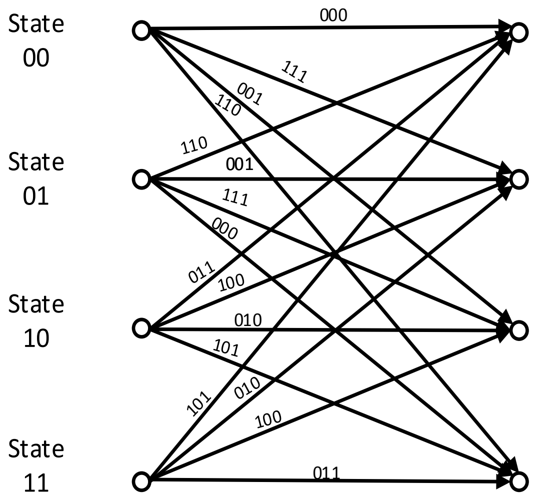


Figure 4-2: Trellis diagram of punctured convolutional code

Figure 4-2 is the trellis representation of a $r = 2/3$ punctured convolutional encoder in which the first output bit of the second interval is punctured using $P = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$ puncturing matrix.

The next step is to prune the desired paths from the code set C that match the codebook B_p generated using extended prefix codes.

Example 2

Consider the codebook designed in Section 4.1.2. In order to search these paths and its corresponding outputs, a Matlab program was simulated that constructed all the possible paths and then pruned the paths that matched the code words of our designed codebook. The steps involved in the Matlab simulation will now be discussed.

The first step is to choose the length M of puncturing matrix P. After a few tries with different values it was found that the value of M should be large enough to produce sufficient paths available for pruning. Therefore, $M = k$ was chosen, therefore $i = 7$ time intervals of the trellis were selected to construct an extended prefix code word of length $n = 8$, as the overall code rate will be $k/(k+1)$.

The second step involves selection of a puncturing pattern for the matrix P. Simulations with all possible valid puncturing patterns were carried out and were able to find enough paths to prune. For this example the puncturing matrix is given by:

$$P = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Where 1 represents the selected bit and 0 represents the punctured bits.

The next step involves generation of all possible paths available from each state to every other state. These paths are then punctured using puncturing matrix P. The total number of paths H produced are calculated by:

$$H = 2^7 \times 2^{(3-1)}$$

$$H = 512$$

In the next step, these 512 paths/code words are compared with the codebook generated in Section 4.1.2. The matched paths with their input sequence and state history were stored in a mapping table that was later used in the simulation. The mapping table below describes the state and input bits for each extended prefix code word of the codebook generated.

Initial State	Input	Next State	Extended Prefix Codebook
0	1101000	0	11100100
0	1101010	1	11100001
0	1101001	2	11100111
0	1101011	3	11100010
1	0111100	0	11100111
1	0111110	1	11100010
1	0111101	2	11100100
1	0111111	3	11100001
2	1000000	0	11100000
2	1000010	1	11100101
2	1000001	2	11100011
2	1000011	3	11100110
3	0010100	0	11100011
3	0010110	1	11100110
3	0010101	2	11100000
3	0010111	3	11100101

Table 4-4: Mapping Table

4.3 Inner Code Construction of the Proposed System

A rate-compatible convolutional encoding system was used as inner code for the proposed scheme. The block diagram of the inner coding scheme is shown as follows:

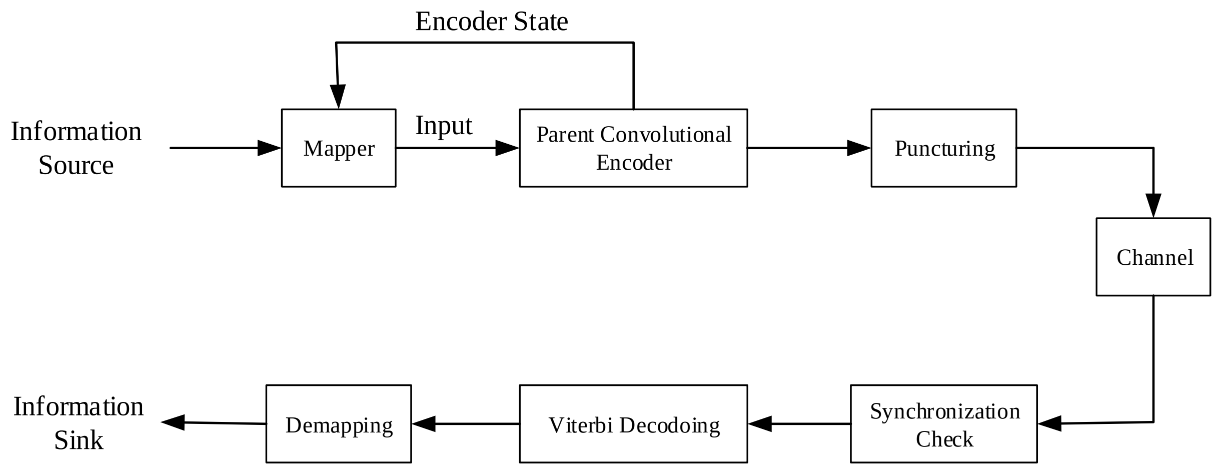


Figure 4-3: Block diagram of Inner Coding scheme

4.3.1 Encoding Process

The information fed into mapper is periodically mapped into a corresponding input sequence depending on the state of the parent encoder, which produces a code word from the codebook Cp and it is transmitted over the channel in front of each frame. The remaining bits of the frame are not mapped to a corresponding input sequence rather normally fed into the encoder.

The pruning period depends on the status of the channel. If the channel is harsh, the pruning period can be decreased to have more synchronization patterns in the transmitted code words. The advantage of the above coding scheme is that the encoding process is a continuous and is not stopped or changed to generate a synchronization pattern before each frame. Each synchronization pattern generated depends on the state of the parent encoder.

4.3.2 Resynchronization

At the receiver end the code word is first passed through the synchronization check process. The resynchronizer looks for the synchronization patterns that were inserted in front of each frame at the encoder through pruning process. As the codebook of these synchronization patterns and the frame length are known to the receiver, it looks for these patterns and their distances between each other. If the distance between patterns is less than the original frame length, that means bits have been lost during the transmission over the channel and deletion errors have occurred. This implies that, a sufficient number of bits are added to make the received code word of length equal to the original frame length. This may introduce substitution errors in the current frame but will resynchronize the subsequent frames. Similarly, when insertion errors occur, the resynchronization process deletes some of the bits from the received code word to make its length equal to the original frame length.

In this research the sliding window method is used to resynchronize the transmission at the receiver. The method works on the fixed sized window that slides over the received bit sequence and looks for the synchronization patterns inserted in front of each frame. It also checks the length of each frame for deletion errors. The resynchronization works for three different cases:

1. When some of the bits from synchronization pattern get deleted.
2. When some of the bits from rest of the frame get deleted.
3. When some of the bits from both synchronization pattern and rest of the frame get deleted.

In case if deletion errors are detected in a frame the resynchronizer calculates the number of deletions and insert required number of bits 0 in front of the frame to make the frame length equal to the actual length of the frame transmitted. This can introduce large burst errors. To improve the performance of the Viterbi decoder the Reed-Solomon code is used in concatenation to the Convolutional code. Interleaving is also introduce to further improve the performance by spreading burst errors apart.

4.3.3 Viterbi Decoding

The Viterbi algorithm is used for decoding the received code words. The received sequence after synchronization check is fed into the Viterbi decoder, which treats this sequence normally and decodes according to the specified puncturing matrix. The advantage of the Viterbi decoder is that it is capable of correcting substitution errors. Therefore, some of the substitution errors caused by resynchronization process are corrected by the decoder. The performance of the decoder also depends on the code rate. For detailed Viterbi decoding algorithm refer to the section 3.5.1 in chapter 3.

4.4 Outer Code Construction of the Proposed System

The proposed coding scheme constitutes of an inner code, as discussed in the previous section and an outer code. A non-binary Reed-Solomon code was used as an outer code. A block diagram of the concatenated system is given below:

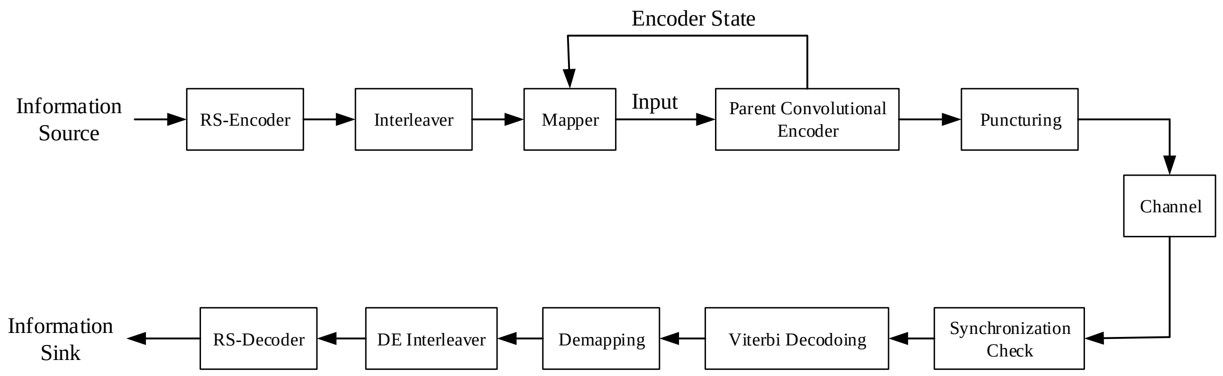


Figure 4-4: Block Diagram of the Concatenated Coding System

4.4.1 Reed-Solomon Code

RS codes are special and a very popular class of non-binary BCH codes that are over $GF\ (q)$, where $q > 2$. Even though RS codes are subclass of BCH codes, they were constructed independently using a different approach by I. Reed and G. Solomon in 1960 [66]. RS codes have high capacity to correct both random and burst errors. They were initially designed for deep-space communication, but they have found several other applications such as in mass storage devices, broadband modems, wireless mobile communications systems and so forth.

A q-ary Reed-Solomon code (RS code) is a q-ary BCH code of length $q-1$ generated by:

$$g(x) = (x - \propto^{a+1})(x - \propto^{a+2}) \ldots (x - \propto^{a+\delta-1}) \qquad (4.3)$$

With $a \geq 0$ and $2 \leq \delta \leq q-1$, where $\propto$ is a primitive element of F_q.

Concatenation of RS code as outer codes with a simple binary codes (convolutional codes) as inner codes provides reliable communication and data storage with reduced decoding complexity and higher error correction capability.

4.4.2 Interleaving

In many communication systems errors occur in burst. Sometimes, these burst errors are long and exceeds the error correction capability of the coding scheme and fails to retrieve the original code word. Interleaving mitigates this problem by changing the positions of each symbol in the code word before transmission and rearranging them at the receiver. This can cause burst errors to spread apart making error correction easier.

According to Ramsey [68] a device that rearranges the ordering of a sequence of symbols in some one-to-one deterministic manner is known an interleaver.

Interleaving is used as an accessory to assist error correction techniques to perform better in worst conditions. To correct burst errors one technique is to place an interleaver between the channel encoder and the channel. This spreads the channel symbols in such a way that symbols of each code word are separated by more than the length of a typical burst of errors, making channel look like a random-error channel to the decoder.

Another type of interleaver is block interleaver linked with block codes. Block interleaver divide symbol sequences into blocks in the form of two dimensional arrays. The symbols are read in row wise and read out column wise [69].

In this research, the information blocks were first encoded using a non-binary $(15, 7)$ Reed-Solomon code by adding parity check bits row wise. These code words were then fed into the convolutional encoder by taking bits column wise. This makes code words to interleave and help improve the performance of the outer code.

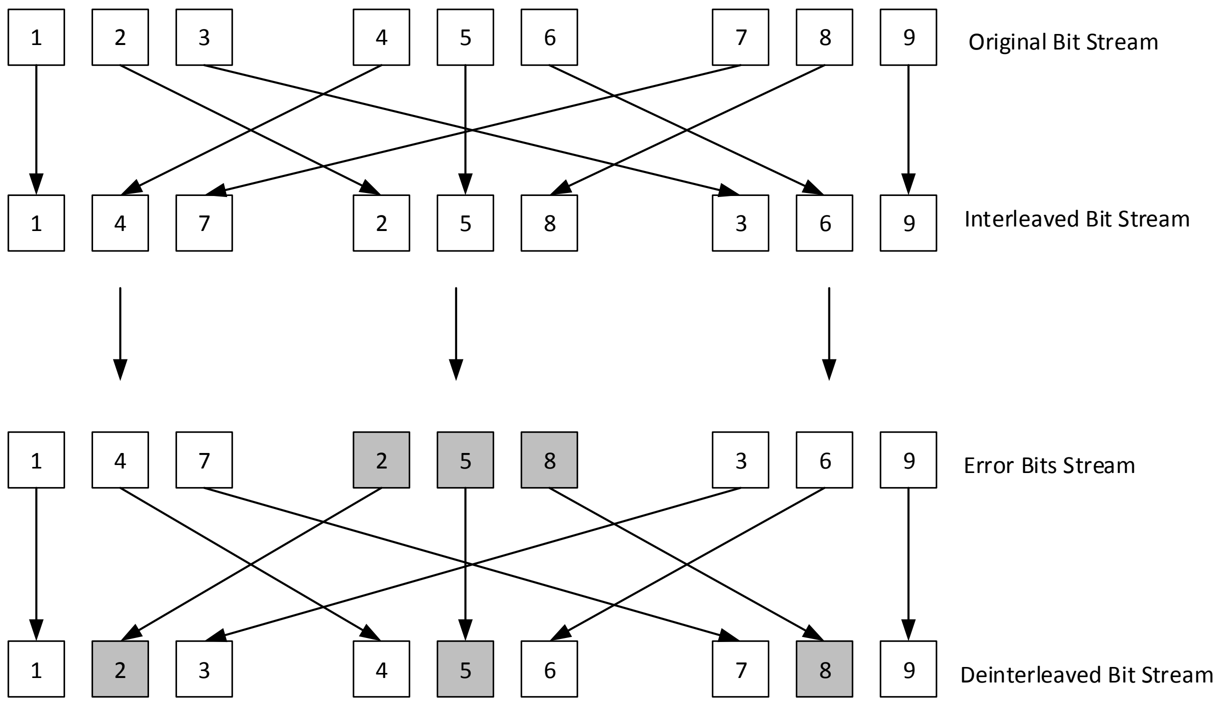


Figure 4-5: The Interleaving Process

Figure 4-5 describes the randomization of burst error; a burst of errors on the channel is converted into isolated errors by using interleaving process.

4.4.3 Reed-Solomon (RS) Decoding

The decoding of non-binary BCH codes (Reed-Solomon codes) is a complex operation and involves more computation than decoding binary BCH codes [67]. It involves the determination of the location and value of errors. Gorenstein and Zierler [70] founded the first decoding procedure for non-binary BCH and RS codes, which was later improved by Chien [71] and Forney [72]. The first efficient decoding algorithm for both binary and non-binary BCH codes was presented by Berlekamp [73]. The Euclidean algorithm can also be used for decoding BCH and Reed-Solomon codes [74]. The Euclidean decoding algorithm is simple and easy to implement. Gore [75] introduced a mechanism to decode BCH and Reed-Solomon code in the frequency domain, which was later modified by Blahut [76] to improve the decoding performance. An overview of the decoding procedure is given below:

Preliminary Decoding Concepts

The received polynomial $R(x)$ can be divided into two parts:

$$R(x) = C(x) + E(x) \tag{4.4}$$

Where $C(x)$ the code word and $E(x)$ is the error polynomial.

Let $E(x) = E_0 + E_1 x + \cdots + E_{s-1} x^{s-1}$ is the expansion of the error polynomial and that there are no more than z errors in the received polynomial. Let $l_1, l_2, l_3, \dots, l_z$ denote the positions of the errors. Therefore each error location l_i is a distinct integer between 0 and $s - 1$.

The error location X_i is defined as $X_i = \alpha^{l_i}$. Hence the error locations $X_1, X_2, \dots, X_z$ are another way of representing the error indices.

Stages of the Decoder:

Figure 4-6 describes the main stages involved in the decoding process. The decoder receives the code word $R(x)$ and outputs the corrected code word $C(x)$.

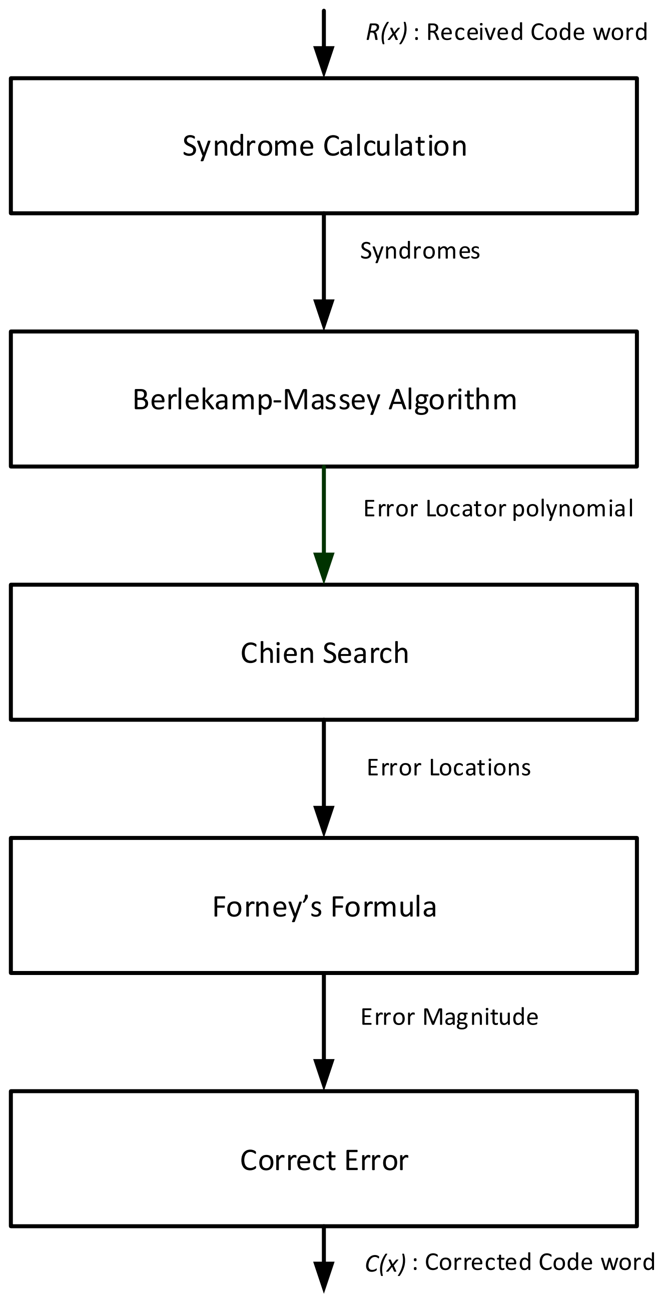


Figure 4-6: Decoding Stages

The steps involved in decoding process are described below [77]:

1. **Syndrome Calculation:**
   The first step involves the calculation of syndromes S_i of the received data, where $i = 1, 2, \ldots, 2t$.
2. **Berlekamp-Massey Algorithm:**
   The second step involves computation of error locator $\sigma(X)$ and error evaluator polynomials $\omega(X)$. After solving for these polynomials the error locations and error magnitudes can be found.
3. **Chien's Search:**
   The next step involves finding the roots of the error locator polynomial. $GF(q^m)$ is a finite field therefore, the idea of Chien's search is to enumerate all the elements of the field to determine the roots. There are some other methods to determine the roots but Chien's search may be the most efficient.
4. **Forney's Formula:**
   After knowing the error locations $\{X_i\}$ the next step is to calculate the error magnitudes $\{Y_i\}$. Chien's algorithm provides the locations of the errors and Forney's formula the magnitude of error at those locations. Thus, the code word $C(x)$ can be calculated using the formula $C(x) = R(x) - E(x)$, and the decoding process is complete.

4.5 Summary

Chapter 4 presented the details of the research work done. The system used path pruned convolutional codes as inner codes and the RS code as an outer code presented as a concatenated coding scheme. This chapter also explained how the codebook was designed and how the synchronization patterns were generated using pruned convolutional codes.

Chapter 5: Computer Simulation and Results

5.0 Introduction

In this chapter, the various aspects of the computer simulation needed in order to evaluate the performance and the effectiveness of the proposed system will be discussed.

Generally, simulations determine efficiency of a coding scheme on the basis of substitution error correction and not how effectively insertion and deletions are detected and corrected. Hence, a conventional bit error rate (BER) versus error probability computer simulation scheme cannot conclusively determine the performance of the proposed concatenated coding system, because the elementary function of the scheme is to detect deletions/insertions and to re-establish the synchronization.

Alternatively, a more complex and detailed simulation is executed by randomly deleting bits from the transmitted code word during the transmission and then at the receiver. The performance is evaluated by resynchronizing and decoding the sequence. By repeating this simulation for several bit deletion probabilities a graph is obtained to show the deletion probability P_d versus bit error rate.

The graphs depict the performance of proposed synchronization error correction scheme using path-pruned convolutional codes in concatenation with RS codes.

5.1 Channel Model

A channel model is required to conduct a simulation. The channel model describes the natural events that can happen during the transmission of a signal on a channel, in this case the occurrence of insertion deletion and substitution errors. The channel model is meant to provide a test bet to measure or estimate the performance of the coding scheme on the basis of parameters extracted from real physical channel.

The channel models that have been used for the insertion/deletion error correction are not generally accepted models, rather they have been designed particularly for the research being carried out. Some of the commonly used insertion deletion channels are Gilbert-Elliot model [15] and Davey-MacKay (DM) binary channel model [12][78][79].

The proposed scheme uses the following simplified channel model based on the binary symmetric channel to carry out the research. Similar model was also used by Swat in [80] and Ferreira dos Santos in [81].

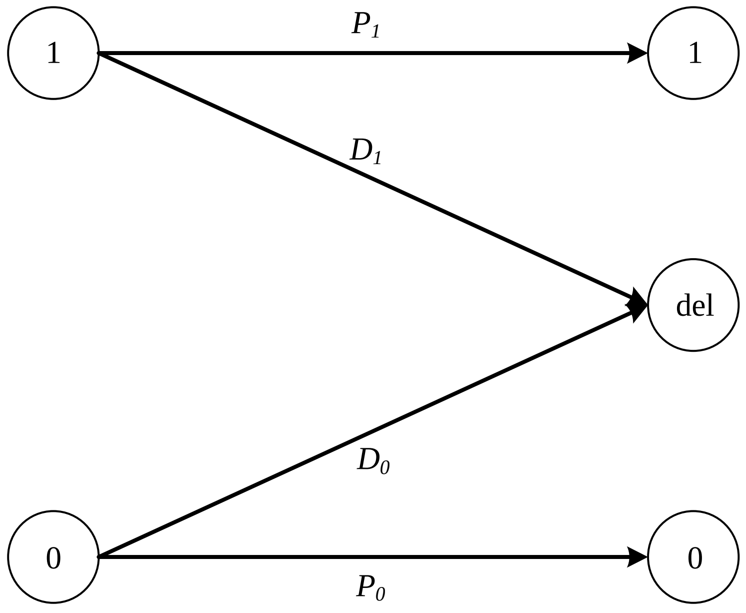


Figure 5-1: Simulation Channel Model

The description of the model parameter is defined in table below.

Parameters	Description
P_0	Probability of receiving 0 when 0 is transmitted
P_1	Probability of receiving 1 when 1 is transmitted
D_0	Probability of not receiving 0 (Deletion probability of 0)
D_1	Probability of not receiving 1 (Deletion probability of 1)

Table 5-1: Channel Model Parameters

The above channel model can be presented by the following equation 5.1:

$$P_0 + P_1 + D_0 + D_1 = 1 \qquad (5.1)$$

Let's consider that when a 0 or a 1 is transmitted over the channel their deletion probabilities are equal, therefore the equation 5.1 can be modified into the following equations 5.2 and 5.3:

$$P_0 = P_1 = P \qquad (5.2)$$

$$D_0 = D_1 = P_d \qquad (5.3)$$

By substituting the values, the equation 5.3 is written as follows:

$$2(P) + 2(P_d) = 1 \qquad (5.4)$$

$$P + P_d = 1/2 \qquad (5.5)$$

This is the simplified form of channel used for the simulation. The probability of the deletion errors P_d that was introduced in the simulation ranged from 10^{-3} to 10^{-4}.

5.2 Simulation Methodology

Random bits are generated at the information source and saved for comparison at the later stages. These randomly generated bits are then encoded using $(15, 7)$ Reed-Solomon code, then interleaved before feeding them to the convolutional encoder. These bits are then divided into small chunks on the basis of pruning period, such that each chunk represents one frame. The first bit of each frame is mapped into a corresponding sequence (which will generate one of the code words from the extended prefix codebook) depending on the state of the convolutional encoder before feeding the whole frame to the convolutional encoder. The convolutional encoder is the normal (n, k, m) encoder with puncturing.

The punctured code word is then transmitted over the channel defined in the channel model section above, which introduces deletion errors in the coded sequence. At the receiver end this sequence is pretreated to look for deletion errors and frames are resynchronized by looking for the predefined extended prefix pattern sequences from codebook. The resynchronized patterns are then fed into the Viterbi decoder which corrects the most substitution errors caused by the resynchronization process. The ability of the decoder to correct substitution errors depends on the free distance of the code. After decoding, the frames are remapped to their corresponding bits, de-interleaved and then fed into the RS decoder. The RS decoder corrects the remaining substitution errors depending upon its error correction capability. The same process is repeated for different deletion error probabilities to evaluate the extent of error correction ability of the concatenated system.

A BER versus deletion probability P_d graph is obtained. This graph compares the randomly generated bits and the decoded bits of the RS decoder. This BER vs. P_d graph evaluates the overall performance of the concatenated coding scheme by comparing the data bits from the information source at the transmitter and the bits decoded by the decoder at the receiver.

Similarly a BER versus deletion probability P_d graph is obtained for the inner coding system which only involves the convolutional encoder and Viterbi decoder. By comparing the two graphs, the effectiveness of the RS codes is shown.

The same process is repeated for different scenarios which include the change in the length of each code word of the extended prefix codebook, change in the conditions of the channel and by changing the code rate and constraint length of the parent encoder.

A RS $(15, 7)$ Reed-Solomon code with $GF(2^4)$ was implemented using Matlab's built in functions 'rsenc' for encoder and 'rsdec' for decoder. This RS code can correct 4 errors within each 15 symbol RS code word and require less processing and time to be implemented for simulation. Commercially a more powerful RS (255.223) code is used [82].

A deletion probability P_d ranged from 10^{-3} to 10^{-4} was used for this simulation. The deletion errors caused by the deletion probability were random through the transmitted code word. The encoding process of the convolutional encoder was continuous and there were no modifications made to the parent encoder. This is considered to be one of the advantages of this proposed concatenated coding scheme, where the pruning process is introduced without stopping or changing the convolutional encoder structure. The set of extended prefix code words and the frame length were known to both the transmitter and the receiver. The frame length can be varied on the base of channel conditions. The better the channel conditions, the lesser is the need for synchronization sequence, therefore the pruning period will increase making the frame length larger. The BER was calculated by comparing the source information and the decoded information at the decoder using:

$$\text{BER} = N_{error}/N_{bits} \tag{5.6}$$

This process was repeated several times for each deletion probability with a very long randomly generated binary information sequence to get a dependable approximated assessment of the system.

5.3 Simulation Results and Discussion

In this section the results of the simulations that were introduced in the previous question will be presented, compared and discussed

The simulations was carried out using different scenarios on the basis of the coding system. These Scenarios include the following:

1. Concatenated coding scheme (with RS code).
2. Different extended prefix code word lengths.
3. Inner coding scheme (without RS code).
4. Different constraint lengths.

A rate $r = 1/2$ parent convolutional encoder with two different constraint length; $K = 3$ & $K = 5$ and two different extended prefix code word lengths (i.e. $L = 8$ & $L = 16$) to perform the simulation experiments of proposed coding scheme for the above mentioned scenarios. Each case will be individually discussed.

Comparison of Concatenated coding scheme vs. Inner Coding Scheme:

Figure 5-2 illustrates the performance of the concatenated coding scheme for synchronization error correction on the basis of BER vs. P_d. Figure 5-2 presents the results of above mentioned all four scenarios for the concatenated coding system. The outer code used is a non-binary Reed-Solomon code RS (15, 7) with $GF(2^4)$. The graph is drawn between deletion probabilities P_d and the Bit Error Rate (BER). The code words transmitted over the channel were long enough to introduce an average of 50 random deletions at each deletion probability. There were no guard spaces or substitution errors introduced during the transmission. The deletions were totally random and there were no restrictions on the number of consecutive bits to be deleted.

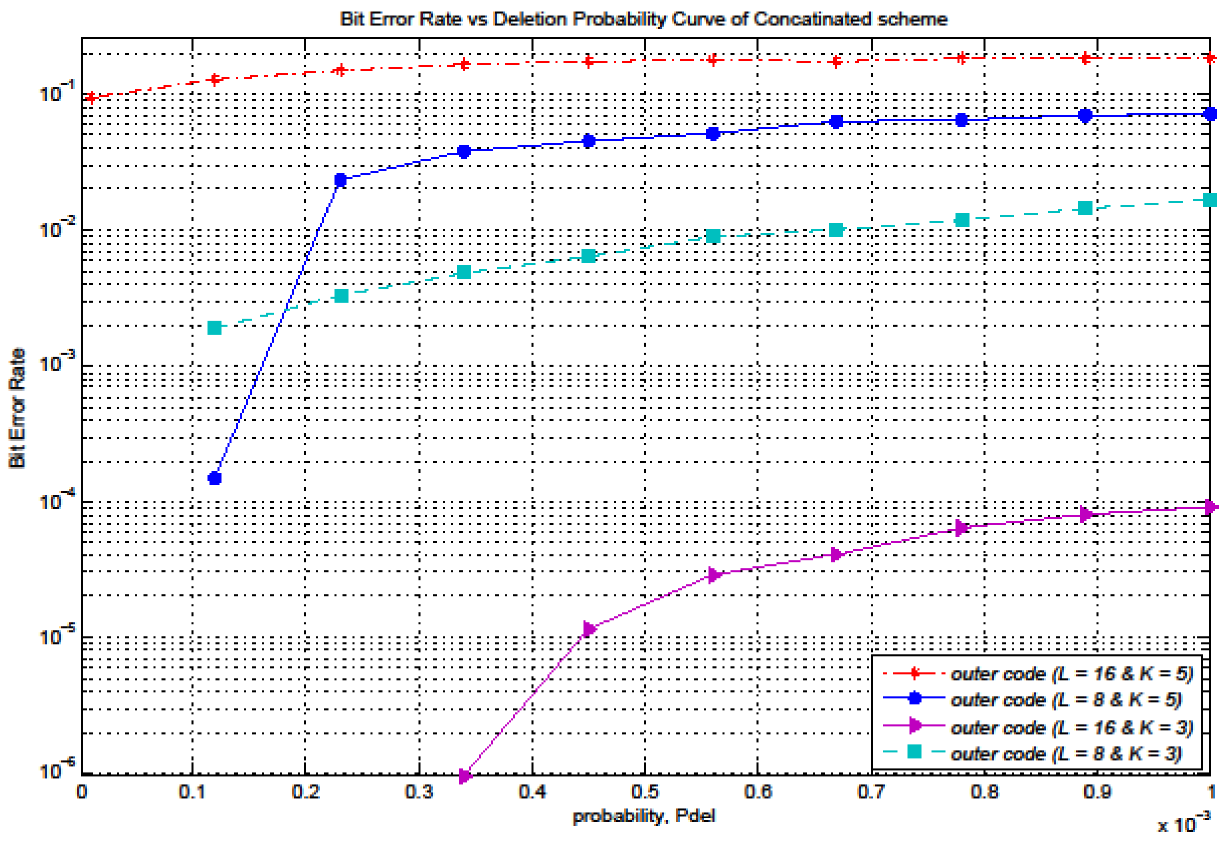


Figure 5-2: BER vs. P_d for $r = 1/2$ convolutional code, with outer RS codes

It can be seen from Figure 5-2 that the concatenated scheme with $L = 16$ & $K = 3$ outperformed all other coding scenarios shown. The system with $L = 16$ & $K = 5$ performed the worst out of all.

The performance of the coding scheme depends on the number of unconstraint positions of the extended prefix that was chosen. The larger the unconstraint part the higher the chances of introducing long bursts of substitution errors after resynchronization. With $L = 16$ & $K = 3$ the constraint part is very long which helps in keeping a better synchronization and hence fewer substitution errors are produced.

If the code with $L = 16\,\&\,K = 3$ is compared to that of code with $L = 8\,\&\,K = 3$ the resynchronization process is better with longer extended prefix lengths having shorter unconstraint parts. This is because the code set is very large (i.e. 2^{30} code words produced at each state of the convolutional encoder is very large as compare to 2^{3} code words in the extended prefix codebook used as synchronization patterns) and very little chances are that synchronization pattern i.e. extended prefix code set will appear in the message portion of the frame. Whereas, in case of shorter extended prefix lengths the code set is not very large (i.e. 2^{14} code words produced at each state of the convolutional encoder is not that large as compare to 2^{3} code words in the extended prefix codebook used as synchronization patterns).

It is better to use large extended prefix codes with shorter unconstraint parts as it will produce larger code sets at the convolutional encoder hence the probability of repeating synchronization patterns in the message part of the fame is less, and even if a deletion occurs during the transmission the chances of producing large substitution errors in a frame during resynchronization is also less. The longer the marker sequence of the extended prefix the better the result will be.

Figure 5-3 shows the BER vs. P_d for the inner coding system only. It can be seen that bit error rate without the outer coding system is higher than what it should be. The reason behind that is the coding rate that is being restricted. This restriction is because of the fact that it needs to have enough branches in the trellis of the punctured code that matches our extended prefix codebook. Because of high rate, the performance of the Viterbi decoder is affected and are not able to correct enough errors as it should. Therefore, the introduction of outer code RS code was essential to correct most of the remaining errors after Viterbi decoding (inner decoder).

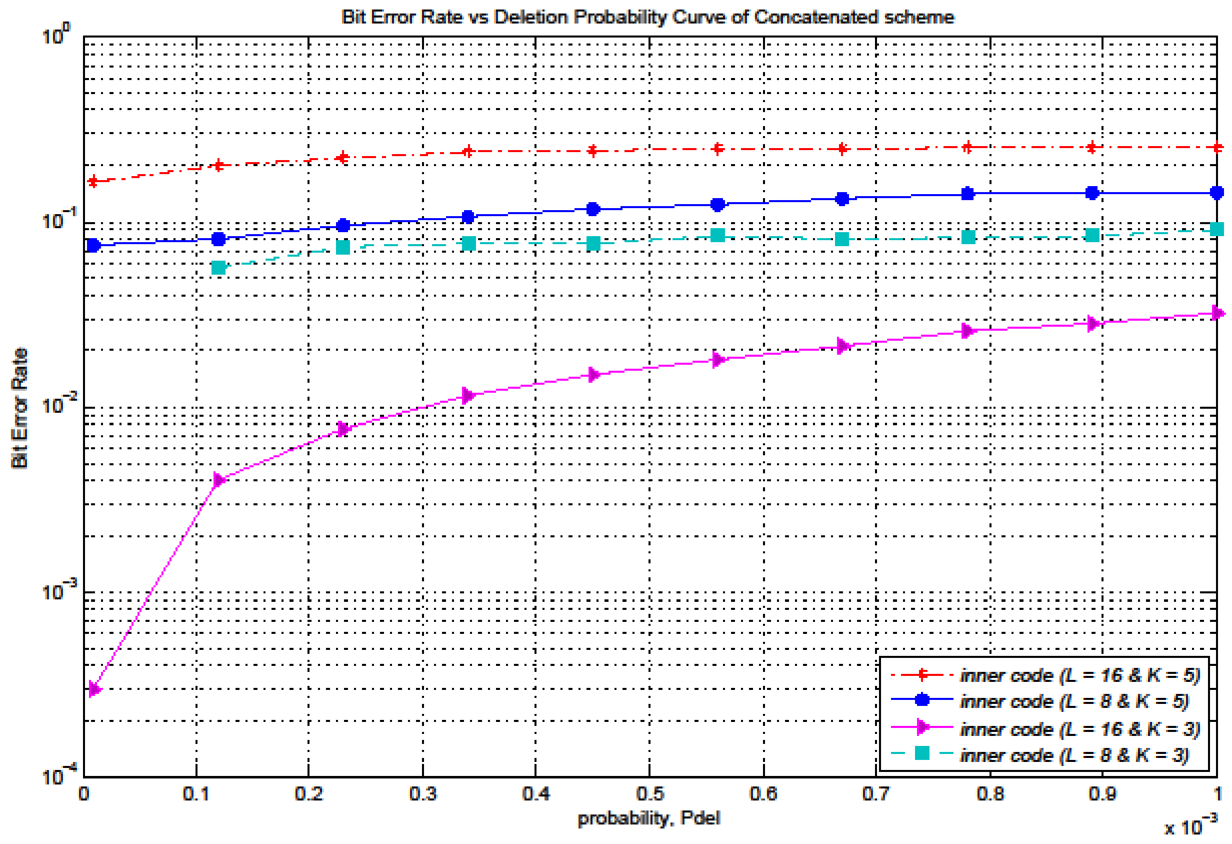


Figure 5-3: BER vs. P_d for $r = 1/2$ convolutional code, without outer RS codes

Comparison of Concatenated coding scheme (with RS code) between different Extended Prefix lengths having same Constraint length:

The next two Figures compare the performance of the two concatenated coding systems with same constraint lengths K but different extended prefix lengths.

Figure 5-4 shows the coding performance of the parent code with constraint length $K = 3$, therefore, both the codes have same unconstraint length of the extended prefix patterns, but the markers they have used are different. It can be seen that the longer extended prefix outperformed the shorter one at a lower constraint length.

Figure 5-5 shows the coding performance of the parent code with constraint length $K = 5$. Since both the codes have the same unconstraint portion but it's larger than what was discussed in the previous Figure 5-4 with constraint length $K = 3$. With larger constraint length the extended prefix codebook size also increases, hence, making synchronization process harder. Also the larger unconstraint parts produce longer synchronization error burst which makes the decoding inefficient.

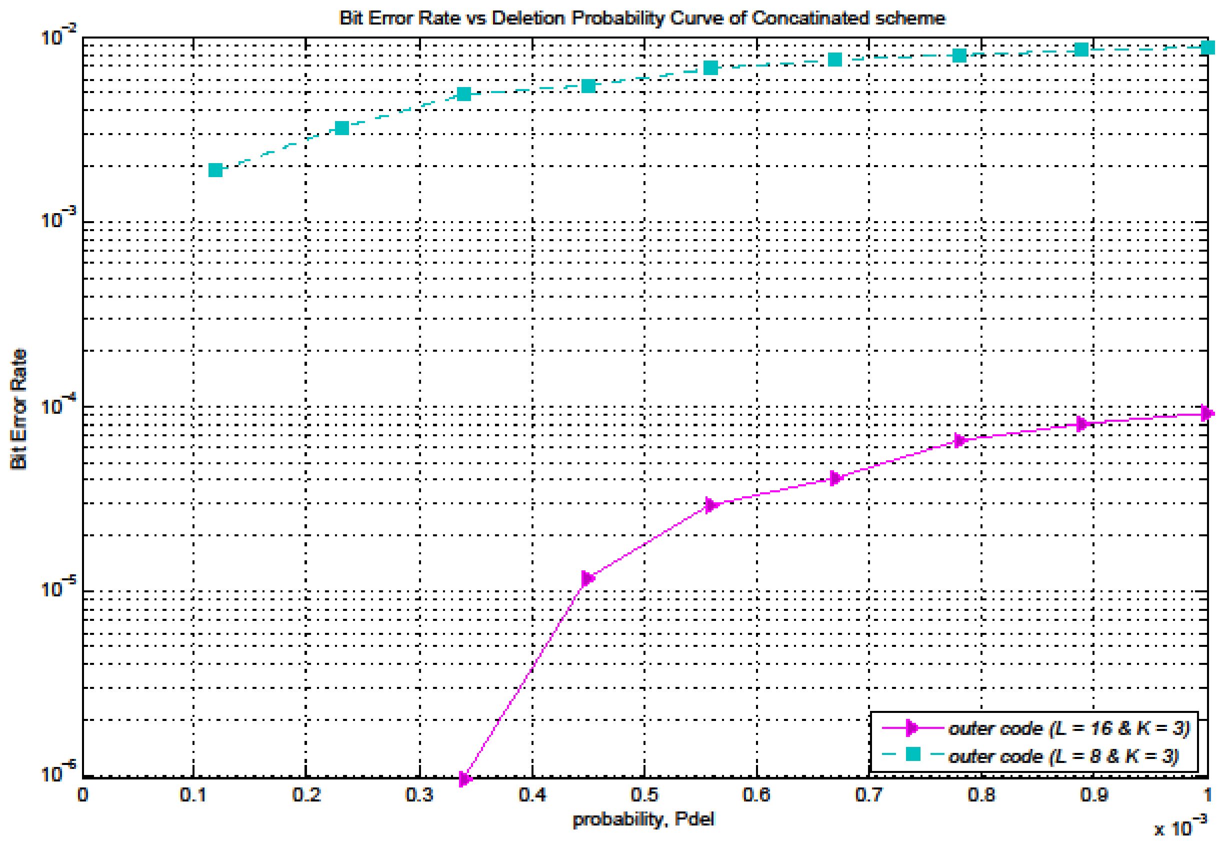


Figure 5-4: BER vs. P_d for $r = 1/2$ & $K = 3$ convolutional code, with outer RS codes

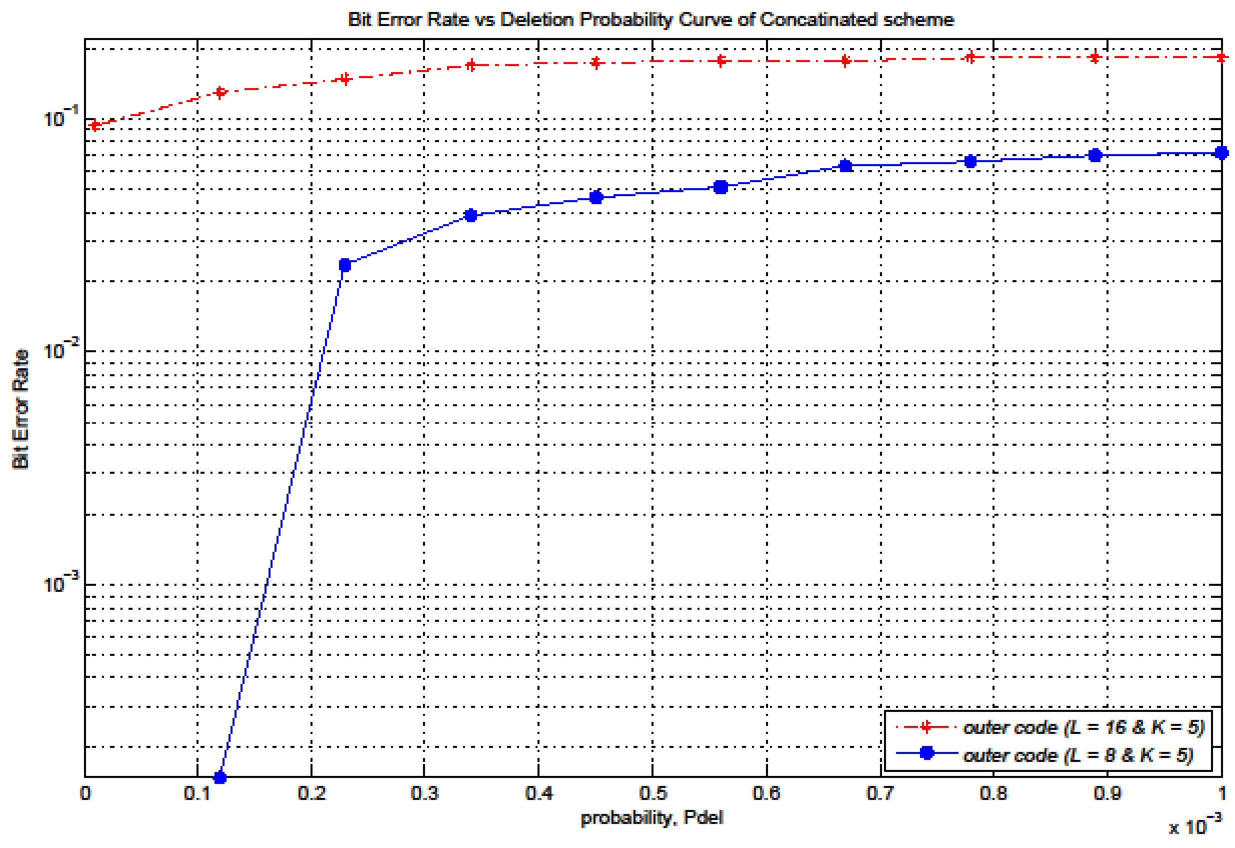


Figure 5-5: BER vs. P_d for $r = 1/2$ & $K = 5$ convolutional code, with outer RS codes

Comparison of inner coding scheme (without RS code) between different Extended Prefix lengths having same Constraint length:

The next two Figures describe the inner code performance with same constraint length but different extended prefix code word lengths.

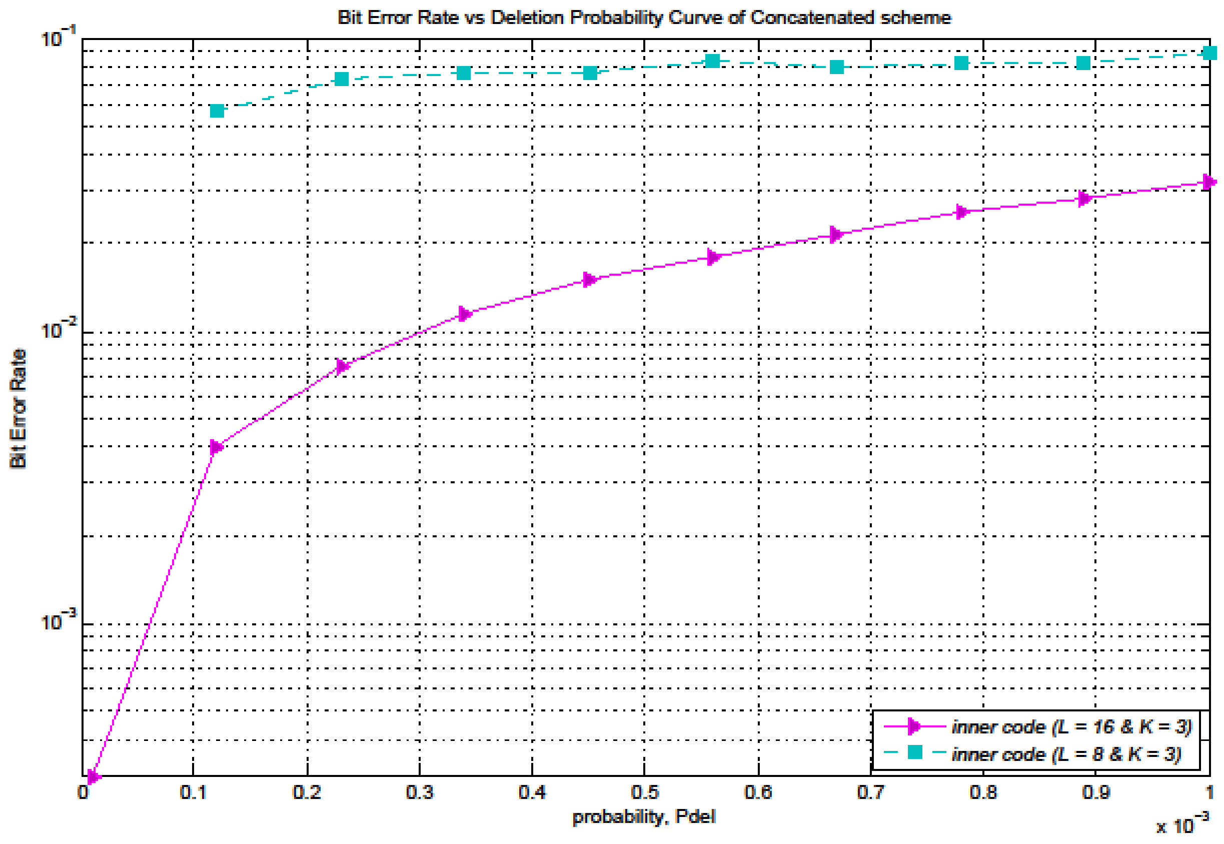


Figure 5-6: BER vs. P_d for $r = 1/2$ & $K = 3$ convolutional code, without outer RS codes

Figure 5-6 shows that the coding scheme with only convolutional encoder and decoder performance is not as good as was expected. It was realized that the performance of the Viterbi decoder depends on the punctured code rate. The high code rate was used due to the constraint of finding enough paths hence the performance of the decoder was degraded.

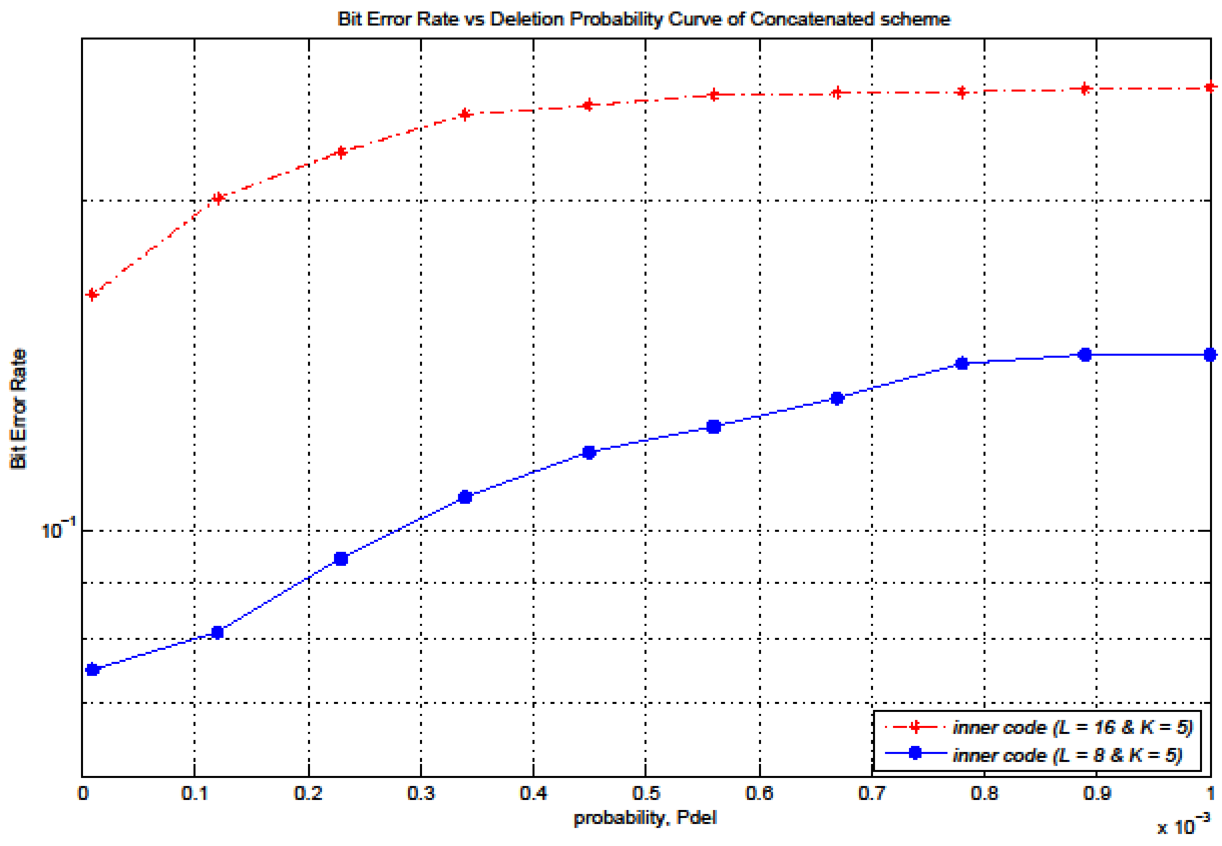


Figure 5-7: BER vs. P_d for $r = 1/2$ & $K = 5$ convolutional code, without outer RS codes

Similarly the performance of inner code by using different length extended prefix codes with constraint length $K = 5$ is shown in Figure 5-7. In this case shorter length extended prefix codes performed better than the longer ones because the unconstraint part of the extended prefix is larger hence, chances of substitution errors are high. Therefore, when resynchronizing a frame the longer length codes will produce more substitution errors.

Comparison of Concatenated coding scheme (with RS code) between same Extended Prefix lengths having different Constraint length:

The last comparison is shown by using same length extended prefix but with different constraint lengths. This means that the constraint part of the extended prefix is same for both constraint lengths.

Figure 5-8 illustrate the performance of the concatenated system for L = 16. The constraint length of the original code was varied and found that longer extended prefix codes perform better with the lower constraint length parent convolutional codes.

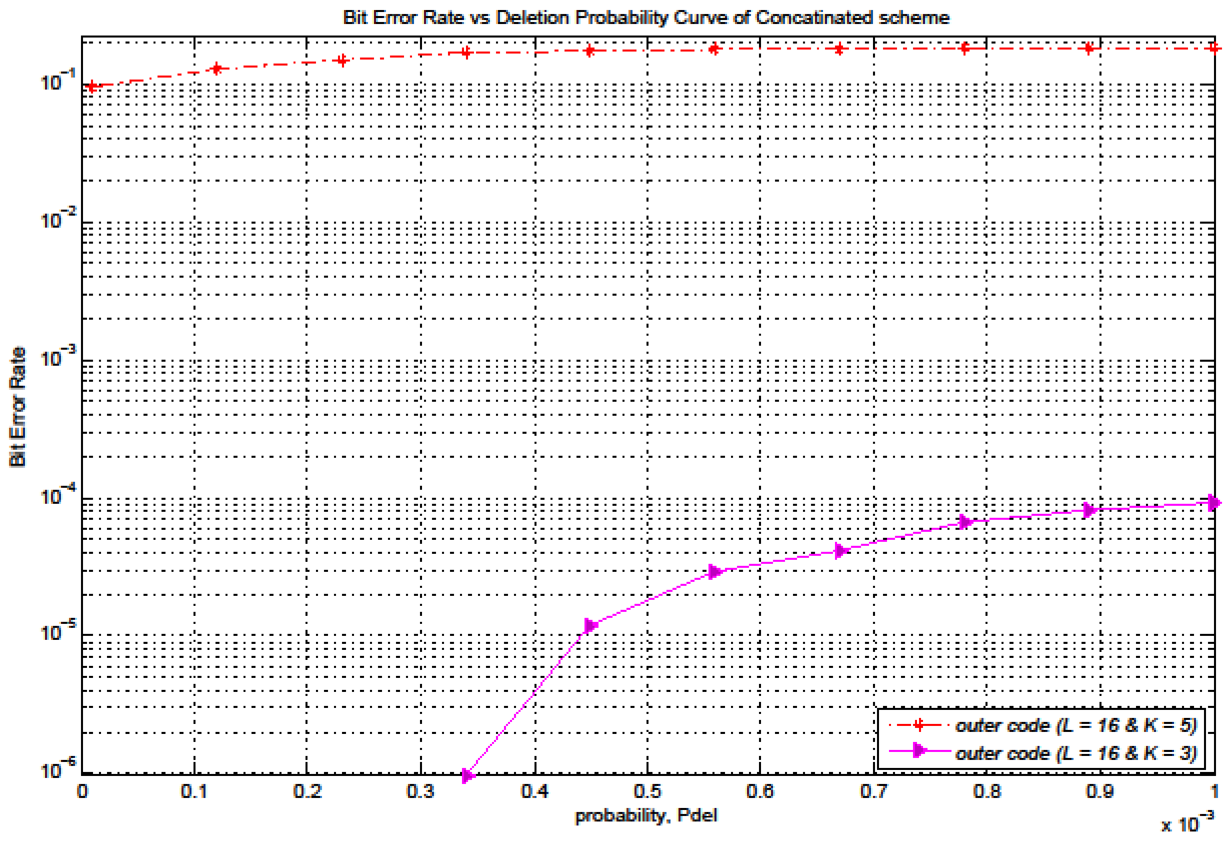


Figure 5-8: BER vs. P_d for $r = 1/2$ & $L = 16$ convolutional code, with outer RS codes

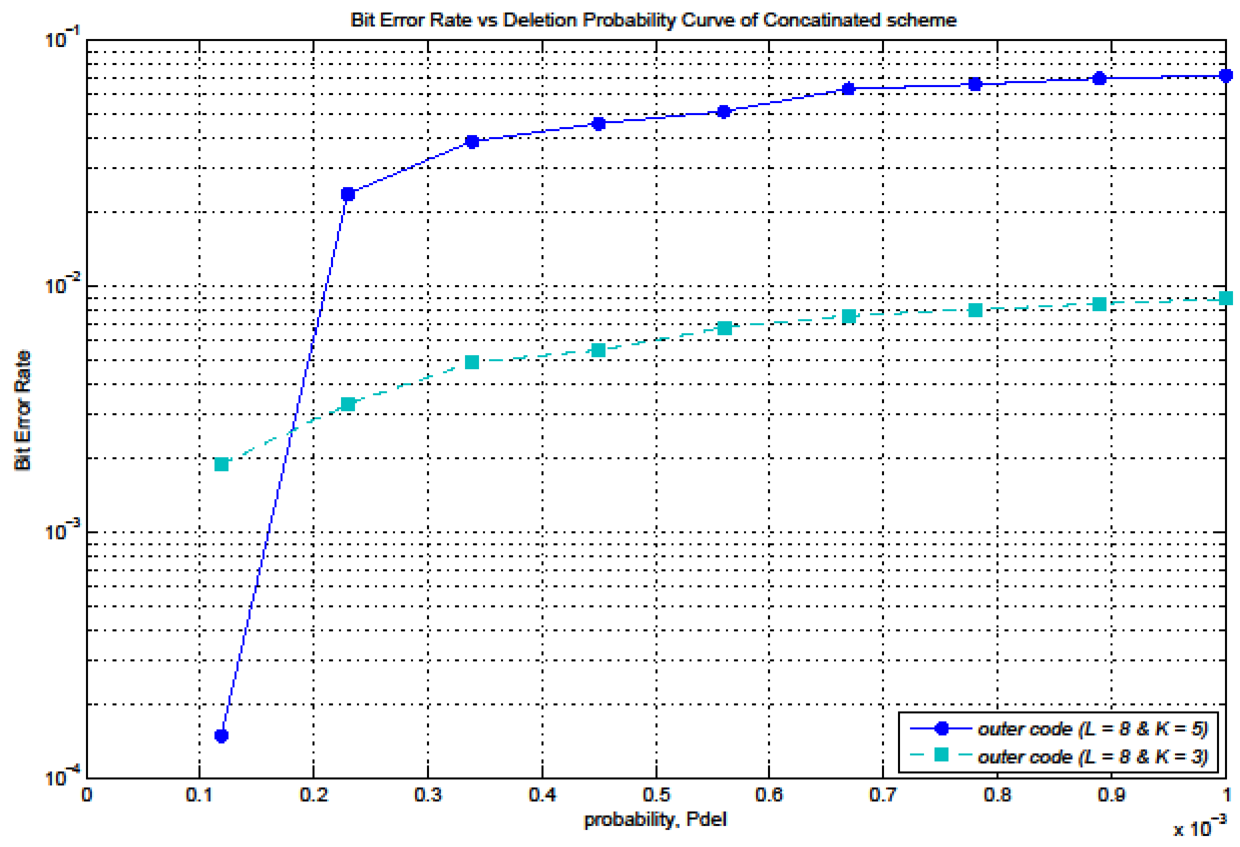


Figure 5-9: BER vs. P_d for $r = 1/2$ & $L = 8$ convolutional code, with outer RS codes

Similarly shorter length extended prefix codes also performs better at lower parent convolutional code constraint lengths. Because with the same extended prefix length the unconstraint part of the code increases by increasing the constraint length which also decreases the marker length in the extended prefix.

5.4 Summary

In this chapter, firstly the channel model and the simulation methodology were presented. Secondly the simulation results were presented and discussed in a methodical manner.

Chapter 6: Research Summary and Conclusion

6.0 Introduction

This chapter provides a brief summary of this research report. Section 6.1 presents a chapter by chapter summary of the research discussed in this research report. Section 6.2 presents conclusion of the research carried out. In section 6.3 some future aspects of the research will be discussed.

6.1 Research Summary

The first chapter defines the problem statement of the research study carried out and constructs the environment in which the research is been conducted.

In Chapter 2, a literature review of the concerned work was introduced in which the channel model, error correction codes and synchronization error correction codes were discussed in detail. These topics are the core of the research and directly linked to it.

In Chapter 3, a background of the techniques was created that were used to accomplish this research. An overview of the synchronization error channel model, convolutional codes, the encoding and decoding of convolutional codes were provided in detail. The puncturing and path pruning techniques were also introduced and discussed and were used for the construction of codebook.

In Chapter 4, the proposed coding scheme that uses path pruned convolutional codes and extended prefix codes were presented. A detailed methodology to design and construct the codebook that uses extended prefix codes and marker sequence was also presented in this chapter. The whole methodology was explained and comprehended with the aid of examples which clarified the whole process step by step. These examples were extracted from the results of the research simulations.

In Chapter 5, the focus was on simulation results. The channel model and the simulation methodology were also presented. The results of these simulations were presented for different scenarios that included concatenated coding scheme with Reed-Solomon codes, inner coding scheme (without RS codes), different extended prefix lengths of the codebook designed and different constraint lengths of the original code.

6.2 Conclusion

A novel coding approach to correct insertion/deletion errors based on rate-compatible convolutional codes and extended prefix codes was presented in this research report. The codebooks were designed using the concepts of extended prefix codes. Each code word in the codebook comprised of a constraint (marker sequence) and an unconstraint part. This results in a set of code words that can be used as synchronization patterns instead of using a single sequence. These code words were then periodically transmitted during the transmission.

The conventional convolutional codes were punctured and path-pruned in this research. The coding scheme used consists of a feedback mapper that lies just before the convolutional encoder. The state of the encoder was fed back to the mapper which decides/maps the information data into input bits to the encoder. The coding process was a continuous process in which the original rate convolutional code was punctured at a certain rate. This puncturing rate was same as that was used in searching the codebook and its corresponding inputs to create the mapping table. The pruning process was introduced periodically by mapping the first bit of the data information frame into a corresponding input to the encoder by feeding its state back to the mapper. Pruning was followed by the normal encoding process with puncturing for the rest of the frame. Hence each frame consisted of an extended prefix code in front and the rest was the normal encoded code word. Therefore the frame was distinguished by looking for these extended prefix at the front of each frame at the receiver to keep synchronization.

After resynchronization the Viterbi decoder decodes these synchronized frames. During the resynchronization process the receiver adds or removes bits when the deletion or insertion error occurs respectively. This introduces substitution errors to the received code word hence the decoding performance of the Viterbi decoder was affected. To improve the performance of the system a concatenated Reed-Solomon code as outer code and interleaving was introduced in the system.

Different simulations were carried out to evaluate the performance of the new proposed coding system by varying different parameters in the system. A simplified binary symmetric channel was used in the simulation experiments to introduce insertion/deletion errors. The designed concatenated coding scheme successfully resynchronized the frames at the receiver and corrected majority of the substitution errors caused due to the resynchronization process.

The system was tested for deletion errors with a rate $r = 1/2$ parent convolutional encoder. The simulation was designed for four different scenarios having two different constraint length $K = 3$ & $K = 5$ and two different extended prefix code word lengths (i.e. extended prefix lengths of 8 & 16). The results showed that the coding system performed better at lower constraint lengths and extended prefix with longer constraint part (marker sequence). The reason it performed better was because at lower constraint lengths the unconstraint part of the extended prefix is shorter. During the resynchronization at the receiver the extended prefix with larger unconstraint part produced longer bursts of substitution errors as compared to the extended prefix with shorter unconstraint part. Because the size of unconstraint part of the extended prefix designed depends on the constraint length of the original code, therefore lower constraint lengths outperformed the higher constraint lengths. By adding Reed-Solomon code as outer coding scheme in concatenation with the new coding scheme the results were improved to greater extent.

To the author's knowledge, this is the first implementation of insertion/deletion error correction coding system using existing path-pruned convolutional codes and extended prefix codes. The performance of this new concatenated scheme showed a great prospect and more developments and improvements are possible in this scheme.

6.3 Future Recommendations

Some possible improvements and developments related to the new coding scheme are now presented:

The higher rate convolutional codes as parent code have not been considered with respect to the new coding scheme. The simulations conducted were for $r = 1/2$ rate convolutional code. Therefore considering higher rates e.g. $r = 2/3, 3/4$ as parent codes may further improve the performance of the system.

By lowering the puncturing rate and finding an optimum extended prefix codebook can also improve the performance of the Viterbi decoder. The Viterbi decoder will be able to correct higher number of substitution errors with more information available and lower puncturing rates.

References

[1] C. E. Shannon, “A mathematical theory of communication,” Bell Sys.Tech. J., vol. 27, 1948, pp. 379–423 and 623–656.

[2] R. W. Hamming, “Error detecting and correcting codes,” Bell Sys. Tech.J. vol. 29, 1950, pp. 147–160,

[3] M. J. E. Golay, “Notes on digital coding,” Proc. IEEE, vol. 37, 1949, pp. 657.

[4] A. Dholakia, “Introduction to convolutional codes with applications.” 1994 by Kluwer Academic Publishers, Boston. ISBN: 0-7923-9467-4.

[5] A. Hocquenghem, “Codes correcteurs d’erreurs,” Chiffres, vol. 2, 1959, pp. 147–156.

[6] B. Sklar, "Digital communications fundamentals and applications," 1988 by Prentice Hall, Inc. Englewood Cliffs, New Jersey, ISBN 0-13-211939-0.

[7] P. Elias, “Predictive coding--I," Information Theory, IRE Transactions on, vol.1, no.1, March 1955, pp. 16-24.

[8] P. Elias,"Predictive coding--II," Information Theory, IRE Transactions on, vol.1, no.1, March 1955, pp. 24-33.

[9] P. Elias, “Coding for noisy channels,” IRE Conv. Record, vol. 4, 1955, pp. 37– 47.

[10] I. S. Reed and G. Solomon, “Polynomial codes over cerain finite fields,” SIAM Journal on Applied Mathematics, vol. 8, 1960, pp. 300–304.

[11] C. Berrou, A. Glavieux, and P. Thitimasjshima, “Near Shannon limit error-correcting coding and decoding: Turbo-codes (1),” in Proc., IEEE Int. Conf. on Commun. (Geneva, Switzerland), May 1993, pp. 1064–1070.

[12] M.C. Davey and D. J C MacKay, "Reliable communication over channels with insertions, deletions, and substitutions," IEEE Transactions on Information Theory, vol.47, no.2, Feb 2001, pp. 687-698.

[13] V. I. Levenshtein, "Binary codes capable of correcting spurious insertions and deletions of ones," Problemy Peredachi Informatsii, vol. 1, no.1, 1965, pp. 12-25.

[14] F., Sellers, Jr., "Bit loss and gain correction code," Information Theory, IRE Transactions on, vol.8, no.1, January 1962, pp. 35-38.

[15] E., Gilbert, "Synchronization of binary messages," Information Theory, IRE Transactions on, vol.6, no.4, September 1960, pp. 470-477.

[16] T.G. Swart, H.C. Ferreira, "Insertion/deletion correcting coding schemes based on convolution coding," Electronics Letters, vol.38, no.16, 1 Aug 2002, pp. 871-873.

[17] M. P F. dos Santos, W.A. Clarke, H.C. Ferreira, T. G. Swart, "Correction of insertions/deletions using standard convolutional codes and the Viterbi decoding algorithm," Information Theory Workshop, 2003. Proceedings. 2003 IEEE, vol., no., 31 March-4 April 2003, pp. 187-190.

[18] L Cheng, H.C. Ferreira, "Rate-compatible path-pruned convolutional codes and their applications on channels with insertion, deletion and substitution errors," Information Theory Workshop, 2005 IEEE, vol., no., 29 Aug.-1 Sept. 2005, pp. 6.

[19] C. Pimentel, I.F. Blake, “Modeling burst channels using partitioned Fritchman's Markov models,” 1998. IEEE, Vehicular technology Transaction, Volume: 47, pp. 885-899.

[20] K. Sh. Zigangirov, “Sequential decoding for a binary channel with drop-outs and insertions,” Probl. Pered. Inform, vol. 5, no. 2, 1969, pp. 23–30.

[21] Wu. Tong, M.A. Armand, “The Davey-MacKay coding scheme for channels with dependent insertion, deletion, and substitution errors,” Magnetics, IEEE Transactions on Volume: 49, Issue: 1, Part: 3 Publication Year: 2013, pp. 489-495.

[22] R. G. Gallager,"Sequential decoding for binary channels with noise and synchronization errors," Massachusetts Inst. of Tech. Lexington Lincoln Lab, Tech. Rep. 2502, Oct. 27th, 1961.

[23] R. W. Lucky, Silicon Dreams: Information, Man, and Machine. New York, NY: St. Martin’s Press, 1989.

[24] S. Wicker, “Error control systems for digital communications and storage,” Englewood Cliffs, NJ: Prentice Hall, Inc., 1995.

[25] D. E. Muller, “Application of boolean algebra to switching circuit design,” IEEE Trans. on Computers, vol. 3, Sept. 1954, pp. 6–12.

[26] E. Prange, “Cyclic error-correcting codes in two symbols,” Tech. Rep. TN-57-103, Air Force Cambridge Research Center, Cambridge, MA, Sept. 1957.

[27] R. C. Bose and D. K. Ray-Chaudhuri, “On a class of error correcting binary group codes,” Information and Control, vol. 3, Mar. 1960, pp. 68–79.

[28] J. M. Wozencraft and B. Reiffen, “Sequential decoding,” Cambridge, MA: MIT Press, 1961.

[29] R. M. Fano, “A heuristic discussion of probabilistic decoding,” IEEE Trans. Inform. Theory, vol. 9, Apr. 1963, pp. 64–74.

[30] F. Jelinek, “An upper bound on moments of sequential decoding effort,” IEEE Trans. Inform. Theory, vol. 15, July 1969, pp. 464–468.

[31] A. J. Viterbi, “Error bounds for convolutional codes and an asymptotically optimum decoding algorithm,” IEEE Trans. Inform. Theory, vol. 13, Apr. 1967, pp. 260–269.

[32] J. L. Ramsey, “Realization of optimum interleavers,” IEEE Trans.Inform. Theory, vol. 16, May 1970, pp. 338–345.

[33] G. D. Forney, “Concatenated codes,” Cambridge, MA: MIT Press, 1966.

[34] S. Benedetto and G. Montorsi, “Serial concatenation of lock and convolutional codes,” Electronics Letters, vol. 32, May 9th 1996, pp. 887–888.

[35] O. Aitsab and R. Pyndiah, “Performance of Reed-Solomon block turbo code,” in Proc., IEEE GLOBECOM, (London, UK), Nov. 1996, pp. 121–125.

[36] J. F. Cheng and R. J. McEliece, “Unit-memory Hamming turbo codes,” in Proc., IEEE Int. Symp. On Inform. Theory, 1995, p. 33.

[37] R. Pyndiah, A. Glavieux, A. Picart, and S. Jacq, “Near optimum decoding of product codes,” in Proc., IEEE GLOBECOM, 1994, pp. 339–343.

[38] M. C. Valenti, “The evolution of error control coding”
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.6249&rep=rep1&type=pdf
Last accessed 12 December 2013.

[39] D. Bardyn, J.A.Briffa, A.Dooms, and P.Schelkens, "Forensic data hiding optimized for JPEG 2000," in Pr oc. IEEE Intern. Symp. on Circuits and Systems, Rio de Janeiro, Brazil, May 15-18, 2011.

[40] J. Hu, T.Duman, E.Kurtas, and M.Erden, "Bit-patterned media with written-in errors: Modeling, detection, and theoretical limits," Magnetics, IEEE Transactions on, vol. 43, no. 8, Aug. 2007, pp. 3517-3524.

[41] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals," Sou. Phys-Dokl, vol.10, no.8, February 1966, pp. 707-710.

[42] B. Gold, "Machine recognition of hand-sent Morse code," Information Theory, IRE Transactions on, vol.5, no.1, March 1959, pp. 17-24.

[43] A. J. Van Wijngaarden, B. Morita, "Extended prefix synchronization codes," Information Theory, 1995. Proceedings, 1995 IEEE International Symposium on, vol., no., 17-22 Sep 1995, pp. 465.

[44] L. J. Guibas, A.M. Odlyzko, “Maximal prefix-synchronized codes”, SIAM J. Appl. Math., vol. 35, no. 2, Sept. 1978, pp. 401-418.

[45] T. G. Swart, H.C. Ferreira, "On multiple insertion/deletion correcting codes," Information Theory, 2000. Proceedings. IEEE International Symposium on, vol., no., 2000, pp.6.

[46] L. Cheng and H. C. Ferreira, "Rate-compatible pruned convolutional codes and Viterbi decoding with the Levenshtein distance metric applied to channels with insertion, deletion and substitution errors," in Proceedings of IEEE AFRICON, Gaborone, Botswana, September 15-17, 2004, pp. 137-143.

[47] L. Cheng and H. C. Ferreira, "Pruned convolutional codes and Viterbi decoding using the Levenshtein distance metric applied to asynchronous noisy channels", Transactions of the SAIEE, vol. 97, no. 2, June, 2006, pp. 140-145.

[48] L. Cheng, H. C. Ferreira and T. G. Swart, "Bidirectional Viterbi decoding using the Levenshtein distance metric for deletion channels," in Proceedings of IEEE Information Theory Workshop, Chengdu, China, October 22-26, 2006, pp. 254-258.

[49] L. Cheng and H. C. Ferreira, "A post-modulation scheme to correct insertion/deletion/substitution errors using the DC2-balanced codes," in Proceedings of IEEE AFRICON, Windhoek, Namibia, Sept. 26-28, 2007, pp. 1-5.

[50] K. A. S. Abdel-Ghaffar, H. C. Ferreira and L. Cheng, "On linear and cyclic codes for correcting deletions," in Proceedings of the IEEE International Symposium on Information Theory, Nice, France, June 24-29, 2007. pp. 851-855.

[51] K. A. S. Abdel-Ghaffar, H. C. Ferreira and L. Cheng, "Correcting deletions using linear and cyclic codes ," IEEE Transactions on Information Theory, vol. 56, no. 10, October 2010, pp. 5223-5234.

[52] H. C. Ferreira, K. A. S. Abdel-Ghaffar, L. Cheng, T. G. Swart and K. Ouahada, "Moment balancing templates: constructions to add insertion/deletion correction capability to error correcting or constrained codes," IEEE Transactions on Information Theory, vol. 55, no. 8, August 2009, pp. 3494-3500.

[53] H. C. Ferreira, K. A. S. Abdel-Ghaffar, L. Cheng and T. G. Swart, "Moment balancing templates: universal constructions to add insertion/deletion correction capability to arbitrary error correcting or constrained codes," in Proceedings of the IEEE International Symposium on Information Theory, Nice, France, June 24-29, 2007. pp. 1676-1680.

[54] D. Slepian, “Permutation modulation,” Proc. IEEE, vol. 53, Mar. 1965, pp. 228-236.

[55] J. G. Dunn, “Coding for continuous sources and channels,” Ph.D. dissertation, Columbia Univ., New York, 1965.

[56] T. Berger, F. Jelinek, and J. K. Wolf, “Permutation codes for sources,” IEEE Trans. Inform. Theory, vol. IT-18, Jan. 1972, pp. 160–169.

[57] T. Berger, “Optimum quantizers and permutation codes,” IEEE Trans. Inform. Theory, vol. IT-18, Nov. 1972, pp. 759–765.

[58] T. Berger, "Minimum entropy quantizers and permutation codes," Information Theory, IEEE Transactions on, vol.28, no.2, Mar 1982, pp. 149-157.

[59] L. Cheng, T. G. Swart and H. C. Ferreira, "Synchronization using insertion/deletion correcting permutation codes," in Proceedings of the IEEE International Symposium on Power Line Communications and its Applications, Jeju City, Jeju Island, South Korea, April 2-4, 2008, pp. 135-140.

[60] L. Cheng, T. G. Swart and H. C. Ferreira, "Re-synchronization of permutation codes with Viterbi-like decoding," in Proceedings of the IEEE International Symposium on Power Line Communications and its Applications, Dresden, Germany, March 29-April 1, 2009, pp. 36-40.

[61] L. Cheng, "Pruned convolutional codes and Viterbi decoding with the Levenshtein distance metric" Master’s degree thesis, Electrical and Electronic Engineering, University of Johannesburg UJ, 2005.

[62] R. J. McEliece, L. Wei, “The trellis complexity of convolutional codes,” 1996. IEEE, Transactions on Information Theory, Volume: 42, pp. 1855 – 1864.

[63] J. Hagenhauer, “Rate-compatible punctured convolutional codes and their applications,” Communications on April 1988. IEEE Transactions, volume 36, pp. 389-400.

[64] B. Brink, H.C. Ferreira, W.A. Clarke, "Pruned convolutional codes for flexible unequal error protection against insertion/deletion/reversal errors," Information Theory, 2000. Proceedings. IEEE International Symposium on, vol., no., 2000, pp. 260.

[65] Jr. G. D. Forney, "The Viterbi algorithm," Proc. IEEE, vol. 61. No. 3, 1973, pp. 268-278.

[66] I. S. Reed, and G. Solomon, “Polynomial codes over certain finite fields,” SIAM J. Applied Math., Vol. 8, 1960, pp. 300–304.

[67] P. Sweeney, "Error control coding from theory to practice," John Wiley & Sons, LTD, 2002.

[68] J. Ramsey, "Realization of optimum interleavers," Information Theory, IEEE Transactions on, vol.16, no.3, May 1970, pp. 338-345.

[69] K. Brayer, O. Cardinale, "Evaluation of Error Correction Block Encoding for High-Speed HF Data," Communication Technology, IEEE Transactions on, vol.15, no.3, June 1967, pp. 371-382.

[70] D. Gorenstein and N. Zierler, "A class of cyclic linear error-correcting codes in pm symbols," J. Soc. Ind. Appl. Math., 9: June 1961, pp. 107-214.

[71] R. T. Chien, "Cyclic decoding procedure for the Bose-Chaudhuri-Hocquenghem codes,: IEEE Trans. Inf. Theory, IT-10: October 1964, pp. 357-363.

[72] G. D. Forney, "On decoding BCH codes," IEEE. Inf. Theory, IT-11: October 1965, pp. 549-557.

[73] E. R. Berlekamp, Algebric Coding theory, McGraw-Hill, New York, 1968.

[74] Y. Sugiyama, M. Kasahara, S. Hirasawa, and T. Namekawa. "A method for solving key equation for decoding Goppa codes," Inf. Contro;, 27: January 1975, pp. 87-99.

[75] W. C. Gore, "Transmitting binarysymbols with Reed-Solomon codes." Proc. Conf. Infor. Sci. and Syst. Princeton, N.J., 1973, pp. 495-497.

[76] R. E. Blahut, "Transfoem Techniques for error-control codes," IBM J. Res. Dec., 23(3), May 1979, pp. 299-315.

[77] T. K. Moon, "Error correction coding," Mathematical Methods and Algorithms. Jhon Wiley and Son, 2005.

[78] E. A. Ratzer, "Marker codes for channels with insertions and deletions," Annals of Telecommunications, 2005.

[79] J. A. Briffa and H.G.Schaathun, "Improvement of the Davey-MacKay construction," in Proc. IEEE Intern. Symp. On Inform. Theory and its Applications, Auckland, New Zealand, Dec. 7-10, 2008, pp. 235-238.

[80] T. G. Swat, “Coding and bounds for correcting insertion/deletion errors,” Master’s degree thesis, Electric and Electronic Engineering, Rand Afrikaans University RAU, 2001.

[81] M. P. Ferreira dos Santos, “Insertion/deletion detection and bit resynchronization using the viterbi algorithm,” Master’s degree thesis, Eletric and Electric Engineering, Rand Afrikaans University RAU, 2003.

[82] Recommendation for space data sytem standards, “Telemetry channel coding,” Blue Book, consultive committie for space data systems meeting, Oberpfaffenhofen, Germany, May 1992.

Appendix A: Some Example Extended Prefix Codes Designed

In this Appendix some extended prefix example codebooks designed in chapter 4 are presented.

Marker Sequence (Constraint Part)	Unconstraint Part	Extended Prefix Code Word
11100	000	11100000
11100	001	11100001
11100	010	11100010
11100	011	11100011
11100	100	11100100
11100	101	11100101
11100	110	11100110
11100	111	11100111

Table A- 1: Extended Prefix Code for Constraint Length $K = 3$ and $L = 8$

Marker Sequence (Constraint Part)	Unconstraint Part	Extended Prefix Code Word
1110	0000	11100000
1110	0001	11100001
1110	0010	11100010
1110	0011	11100011
1110	0100	11100100
1110	0101	11100101
1110	0110	11100110
1110	0111	11100111
1110	1000	11101000
1110	1001	11101001
1110	1010	11101010
1110	1011	11101011
1110	1100	11101100
1110	1101	11101101
1110	1110	11101110
1110	1111	11101111

Table A- 2: Extended Prefix Code for Constraint Length $K = 4$ and $L = 8$

Marker Sequence (Constraint Part)	Unconstraint Part	Extended Prefix Code Word
110	00000	11000000
110	00001	11000001
110	00010	11000010
110	00011	11000011
110	00100	11000100
110	00101	11000101
110	00110	11000110
110	00111	11000111
110	01000	11001000
110	01001	11001001
110	01010	11001010
110	01011	11001011
110	01100	11001100
110	01101	11001101
110	01110	11001110
110	01111	11001111
110	10000	11001000
110	10001	11010001
110	10010	11010010
110	10011	11010011
110	10100	11010100
110	10101	11010101
110	10110	11010110
110	10111	11010111
110	11000	11011000
110	11001	11011001
110	11010	11011010
110	11011	11011011
110	11100	11011100
110	11101	11011101
110	11110	11011110
110	11111	11011111

Table A- 3: Extended Prefix Code for Constraint Length $K = 5$ and $L = 8$

Marker Sequence (Constraint Part)	Unconstraint Part	Extended Prefix Code Word
1111111110000	000	1111111110000000
1111111110000	001	1111111110000001
1111111110000	010	1111111110000010
1111111110000	011	1111111110000011
1111111110000	100	1111111110000100
1111111110000	101	1111111110000101
1111111110000	110	1111111110000110
1111111110000	111	1111111110000111

Table A- 4: Extended Prefix Code for Constraint Length $K = 3$ and $L = 16$

Marker Sequence (Constraint Part)	Unconstraint Part	Extended Prefix Code Word
111111110000	0000	11111111100000000
111111110000	0001	11111111100000001
111111110000	0010	11111111100000010
111111110000	0011	11111111100000011
111111110000	0100	11111111100000100
111111110000	0101	11111111100000101
111111110000	0110	11111111100000110
111111110000	0111	11111111100000111
111111110000	1000	11111111100001000
111111110000	1001	11111111100001001
111111110000	1010	11111111100001010
111111110000	1011	11111111100001011
111111110000	1100	11111111100001100
111111110000	1101	11111111100001101
111111110000	1110	11111111100001110
111111110000	1111	11111111100001111

Table A- 5: Extended Prefix Code for Constraint Length $K = 4$ and $L = 16$

Marker Sequence (Constraint Part)	Unconstraint Part	Extended Prefix Code Word
11111110000	00000	1111111000000000
11111110000	00001	1111111000000001
11111110000	00010	1111111000000010
11111110000	00011	1111111000000011
11111110000	00100	1111111000000100
11111110000	00101	1111111000000101
11111110000	00110	1111111000000110
11111110000	00111	1111111000000111
11111110000	01000	1111111000001000
11111110000	01001	1111111000001001
11111110000	01010	1111111000001010
11111110000	01011	1111111000001011
11111110000	01100	1111111000001100
11111110000	01101	1111111000001101
11111110000	01110	1111111000001110
11111110000	01111	1111111000001111
11111110000	10000	1111111000001000
11111110000	10001	1111111000010001
11111110000	10010	1111111000010010
11111110000	10011	1111111000010011
11111110000	10100	1111111000010100
11111110000	10101	1111111000010101
11111110000	10110	1111111000010110
11111110000	10111	1111111000010111
11111110000	11000	1111111000011000
11111110000	11001	1111111000011001
11111110000	11010	1111111000011010
11111110000	11011	1111111000011011
11111110000	11100	1111111000011100
11111110000	11101	1111111000011101
11111110000	11110	1111111000011110
11111110000	11111	1111111000011111

Table A- 6: Extended Prefix Code for Constraint Length $K = 5$ and $L = 16$

Appendix B: Some Example Mapping Tables

In this Appendix some example mapping tables for some specific parent convolutional codes are presented.

Initial State	Info	Input	Extended Prefix Code Word
0	0	1101000	11100100
0	1	1101011	11100010
1	0	0111100	11100111
1	1	0111111	11100001
2	0	1000000	11100000
2	1	1000011	11100110
3	0	0010100	11100011
3	1	0010111	11100101

Table B- 1: Mapping Table of a Constraint length $K = 3$ Parent Convolutional Encoder and $L = 8$

Initial State	Info	Input	Extended Prefix Code Word
0	0	1001000	11101111
0	1	1001111	11100101
1	0	0101000	11100111
1	1	0101111	11101101
2	0	0011000	11100011
2	1	0011111	11101001
3	0	1111000	11101011
3	1	1111111	11100001
4	0	0000000	11100000
4	1	0000111	11101010
5	0	1100000	11101000
5	1	1100111	11100010
6	0	1010000	11101100
6	1	1010111	11100110
7	0	0110000	11100100
7	1	0110111	11101110

Table B- 2: Mapping Table of a Constraint length $K = 4$ Parent Convolutional Encoder and $L = 8$

Initial State	Info	Input	Extended Prefix Code Word
0	0	1110000	11010111
0	1	1111111	11001111
1	0	0100000	11010100
1	1	0101111	11001100
2	0	1010000	11000011
2	1	1011111	11011011
3	0	0000000	11000000
3	1	0001111	11011000
4	0	0110000	11011111
4	1	0111111	11000111
5	0	1100000	11011100
5	1	1101111	11000100
6	0	0010000	11001011
6	1	0011111	11010011
7	0	1000000	11001000
7	1	1001111	11010000
8	0	1010000	11010011
8	1	1011111	11001011
9	0	0000000	11010000
9	1	0001111	11001000
10	0	1110000	11000111
10	1	1111111	11011111
11	0	0100000	11000100
11	1	0101111	11011100
12	0	0010000	11011011
12	1	0011111	11000011
13	0	1000000	11011000
13	1	1001111	11000000
14	0	0110000	11001111
14	1	0111111	11010111
15	0	1100000	11001100
15	1	1101111	11010100

Table B- 3: Mapping Table of a Constraint length K = 5 Parent Convolutional Encoder and $\boldsymbol{L = 8}$

Appendix C: Matlab Simulation Code

Code to construct transition table used to create all possible code words at each state of the encoder

```
function [Transition_Table] = transition_table
ConstraintLength = 3;
CodeGenerator = [5 7];
Transition_Table = [];
trellis = poly2trellis(ConstraintLength,CodeGenerator);
i_s = 0;

for j = 1:1:length(trellis.outputs)
    Transition_Table = [Transition_Table;  i_s, 0, trellis.nextStates(j,1),
trellis.outputs(j,1); i_s, 1, trellis.nextStates(j,2),
trellis.outputs(j,2)];
    i_s = i_s+1;
end

end
```

Code to search extended prefixes and create mapping table

```
function [Table_Mapping] = marker_search_mapping_table

%  disp('    i_s  input  n_s  output');
Transition_Table = transition_table; % inputs the transition table from the
function

 sequence=[]; % Record the code words generated
 state_hist=[]; % Record the state history for each transition
 input_hist =[]; % Record the input sequence for each code word

 Extended_Prefix = [  1 1 1 0 0 0 0 0;
                      1 1 1 0 0 0 0 1;
                      1 1 1 0 0 0 1 0;
                      1 1 1 0 0 0 1 1;
                      1 1 1 0 0 1 0 0;
                      1 1 1 0 0 1 0 1;
                      1 1 1 0 0 1 1 0;
                      1 1 1 0 0 1 1 1];

 i_s=[];
 n_s=[];

 % loop to generate all possible code words at each state

for r = 1:1:length(Transition_Table)

i_s=Transition_Table(r,1);
n_s=Transition_Table(r,3);
```

```
      [a]= find( Transition_Table(:,1)== n_s);
        for b=1:1:length(a)

         n_s= Transition_Table(a(b),3);

            [c]= find( Transition_Table(:,1)== n_s);
                for d=1:1:length(c)

                   n_s= Transition_Table(c(d),3);

                    [e]= find( Transition_Table(:,1)== n_s);
                        for f=1:1:length(e)

                            n_s= Transition_Table(e(f),3);

                              [g]= find( Transition_Table(:,1)== n_s);
                                  for h=1:1:length(g)

                                      n_s= Transition_Table(g(h),3);

                                        [i]= find( Transition_Table(:,1)==
n_s);
                                            for j=1:1:length(i)

                                                n_s=
Transition_Table(i(j),3);

                                                    [k]= find(
Transition_Table(:,1)== n_s);
                                                        for
lt=1:1:length(k)


sequence=[sequence;Transition_Table(r,4),Transition_Table(a(b),4),Transitio
n_Table(c(d),4),Transition_Table(e(f),4),Transition_Table(g(h),4),Transitio
n_Table(i(j),4),Transition_Table(k(lt),4)]; % creates all possible code
words generated at all states of the encoder for N time intervals

state_hist = [state_hist;Transition_Table(r,1),Transition_Table(k(lt),3)];
% stores the state history for the code word produced for N time intervals

input_hist =
[input_hist;Transition_Table(r,2),Transition_Table(a(b),2),Transition_Table
(c(d),2),Transition_Table(e(f),2),Transition_Table(g(h),2),Transition_Table
(i(j),2),Transition_Table(k(lt),2)]; % stores the input sequence for each
code word produced by the encoder during N time intervals


                                                        end
                                            end

                                  end

                        end

                end
        end



end

sequence;
sequence2 = reshape(sequence.',1,[]);
sequence_dec = sequence2;

sequence_bin = [];
for dec = 1:1:length(sequence_dec);
    sequence_bin = [sequence_bin dec2bin(sequence_dec(1,dec),2)- '0'];
end
input = reshape(sequence_bin.',1,[]);
t1= reshape(input, size(sequence,2)*2 ,
size(input,2)/(size(sequence,2)*2))'; % contains all possible code words at
each state


        t = t1(:,[1 3 5 7 9 11 13 14]); % Puncturing pattern determines the
positions of 1s i.e. bits that are kept

      figure;

hh = [];
in = [];
ot = [];
for e = 1:1:size(Extended_Prefix,1)

    l=
ismember(t(:,1:size(Extended_Prefix,2)),Extended_Prefix(e,:),'rows'); %
find the element of Z (Extended_Prefix) in t (punctured sequence)
    hh= [hh; state_hist([find(l)],:)]; % state history of the found pattern
in t
    in= [in; input_hist([find(l)],:)]; % input sequence of the found
patterns
    ot = [ot; t([find(l)],:)]; % code words found


    hh_r=reshape(hh.',1,[]); % reshaping state history for ploting
    hh_dec = hh_r;



    hhh= reshape(hh_dec, 2,length(hh_dec)/2);


    plot(hhh); % plotting state history
    set(gca,'YDir','reverse');
    xlabel('Trellis');
    ylabel('States');
end
table1 = [];
for z = 1:1:length(in)
```

```
    table1 = [table1; hh(z,1) in(z,:) ot(z,:)];
end

table2 = [];
for z = 1:1:length(in)
    table2 = [table2; hh(z,1) bin2dec(cellstr(sprintf('%d',in(z,:))))];
end

table2 = sortrows(table2);

Table = [];
in_put = 0;
for z = 1:1:length(table2)
    Table = [Table; table2(z,:) in_put];
    in_put = xor(in_put,1);
end
limit = 2^2;
count1 = 1;
count2 = limit;
Table_Mapping = []; % creating mapping table required for the pruning
process

for u = 1:1:length(Table)/limit
    Table_Mapping = [Table_Mapping; Table(count1,:); Table(count2,:)];
    count1 = count1 + limit;
    count2 = count2 + limit;
end
end
```

Simulation code for the concatenated scheme that includes RS code, interleaving, convolutional encoding, puncturing, pruning, resynchronization, Viterbi decoding and RS decoding

```
function [] = simulation
Mapping_Table = marker_search_mapping_table;       %   mapping table for
introduction of pruning. it consists of initial_state input and info.

p_m =[1 1 1 1 1 1 1; 0 0 0 0 0 0 1]; % puncturing matrix

Extended_Prefix = [   1 1 1 0 0 0 0 0;
                      1 1 1 0 0 0 0 1;
                      1 1 1 0 0 0 1 0;
                      1 1 1 0 0 0 1 1;
                      1 1 1 0 0 1 0 0;
                      1 1 1 0 0 1 0 1;
                      1 1 1 0 0 1 1 0;
                      1 1 1 0 0 1 1 1];


probability = linspace(0.00001,0.001,10); % Error Probabilities.
Bit_Error_Rate = [];
Bit_Error_Rate_inner = [];
for prob = 1:1:length(probability) % loop will run for each deletion
probability i.e. 10 time
data_info = [];
op_info =[];
data_info2 = [];
op_info2 =[];
N = 1;

for nn = 1:1:N % loop will run N times to for each deletion probability

 msg_complete = []; % records the complete information message sent
 input2 = []; % records the encoded data after deletion errors
 input3 = []; % recodes the resynchronized code word


% Reed-Solom Encoder

    k = 7;
    m = 4;
    x = [];
    c = [];
    b1 = 616*80; % produce more than 50 thousand bits
     information1=floor(2*rand(1,b1)); % Generate random information bits
equal to the length of b1

     for j = 1:m:length(information1)
         x = [x bin2dec(sprintf('%d',information1(j:j+(m-1))))];
     end
         x = vec2mat(x,k);
         msg_rs = gf(x,m); % Create a Galois array in GF(2^m).
         n = 2^m-1;


         code_rs = rsenc(msg_rs,n,k);
         interleaved = reshape(code_rs,1,[]);
         for r = 1:1:length(interleaved)
             c = [c interleaved(r)];
         end

                    GFInput =[];
                    DecOutput = [];
                    prim_poly = primpoly(m,'nodisplay');
                    information = [];
                    % GFInput = GFInput(:)';% force a row vector
                    GFInput = c;
                    GFRefArray = gf([0:(2^m)-1],m,prim_poly);
                    for i=1:length(GFInput)
                        for k1=0:(2^m)-1
                            temp = isequal(GFInput(i),GFRefArray(k1+1));
                            if (temp==1)
                                DecOutput(i) = k1;
                            end
                        end
                    end

             for u = 1:1:length(DecOutput)
              information = [information dec2bin(DecOutput(u),4) - '0'];
             end



    pruning_period = 21;  % Pruning Period i.e. after every 21 bits 1 bit
is pruned to produce Extended_Prefix for synchronization

    pruning_bit_positions = (1 : pruning_period : length(information));
    count1 = 1; % counter for the bit to be pruned
    count2 = 22;% counter for the bits to be encoded normally
   init_state = 0;




% Mapping

 i_s2 = Mapping_Table(1,1); %   Encoder Starts from all zero state

for j = 1:1:length(information)/22; % loop will run to map the information
                                    % bits according to the pruning period
                                    % and encode the information per frame
                                    % i.e for 22 bits per loop

    info = information(count1); % stores the pruning bit
    info_puncture = information(count1+1:count2); % stores next 21 bits
after pruning bit

    count1 = count1 + 22;
    count2 = count2 + 22;
 % Mapping Process


    info_check= find(Mapping_Table(:,3)==bin2dec(sprintf('%d',info))); %
Checks the location of pruning bit in the Mapping table

    istate2_check= find(Mapping_Table(:,1)==i_s2); % Checks the initial
state which is the final state of the encoder after encoding previous frame

    d = intersect(info_check,istate2_check); % finds the position of the
information bit for pruning from Mapping table

    data1 = dec2bin(Mapping_Table(d,2),7); % Maps 1 bit into 7 bits


data_dec = bin2dec(data1);
data_bin =dec2bin(data_dec,length(p_m))- '0';
msg1 = reshape(data_bin.',1,[]);
msg = [msg1 info_puncture]; % 7 mapped bits plus 21 normal bits makes it 28
msg bits that are encoded by the encoder
msg_complete = [msg_complete msg]; % Complete message bits (mappped and
normal) to check BER over the length of information



% Encoding

ConstraintLength = 3;
CodeGenerator = [5 7];
tblen = 5*ConstraintLength;
opmode = 'trunc';
dectype = 'hard';
puncpat = [1 0 1 0 1 0 1 0 1 0 1 0 1 1]; % Puncturing Matrix
trellis = poly2trellis(ConstraintLength,CodeGenerator); % Trellis formation
[code_conv,final_state] = convenc(msg,trellis,puncpat,init_state); %
Encoding
input2 = [input2 code_conv]; % Code word to be transmitted to the receiver
% i_s2 = str2num(dec2bin(final_state,2));
i_s2 = final_state; % Final state of the Encoder after each frame i.e.
after 22 bits of information or 28 bits of mapped information
% msg_info = [msg_info msg];
init_state = final_state; % Initial state for the encoder to start from
i.e. final state of the previous encoded frame
end



% Introducing deletion error

out = rand(1, length(input2)) <= probability(prob); % Randomly generate the
bits for deletion on the basis of probability

error_pos = find(out==1);% Positions of the bits to be deleted
```

```
input2(:,error_pos) = []; % Code word Received by the Receiver after
deletion of random bits from it on the basis of probability


% Resynchronization

sliding_window =zeros(1,length(input2)); % Generates a sequence of zeros of
the length of received code word


% Creating sliding_window

for i = 1:1:length(input2)-(size(Extended_Prefix,2)-1)
    window = [input2(i:i+(size(Extended_Prefix,2)-1))];
    for s = 1:1:size(Extended_Prefix,1)
        e_w = ismember(window, Extended_Prefix(s,:), 'rows');
        if e_w ==1
            sliding_window (1,i)= 1;
            break;
        else
        end
    end
end

pattern = [1 zeros(1,31)]; % Generates the correct sliding window pattern
of an error free frame

sliding_window = [sliding_window pattern]; % concatenate the correct
pattern of the sliding window at the end of the generated sliding window

find1 = find(sliding_window); % Finds the positions of 1s in the sliding
window (1 shows the starting point of the frame)

temp_vector = []; % to store the unmatched pattern in the sliding window
ref_point = find1(1);


if find1(1) ~= 1; % Case when one of the synchronization bits get deleted
and its the 1st frame of the transmission

    sliding_window = [pattern sliding_window]; % concatenate the correct
pattern of the sliding window at the start of the generated sliding window
for reference

    find1 = find(sliding_window); % Finds the positions of 1s in the
sliding window (1 shows the starting point of the frame)

    ref_point = find1(1); % Reference point for resynchronization purpose

    for i = 1+1:1:length(find1); % Loop through the sliding window to check
for deletions

        if i == length(find1); % case when its the last frame of the
received code word and the synchronization patterns occurs within the
message sequence
```

```
        temp_vector = [sliding_window(ref_point:find1(i)-1)]; % to store
the unmatched pattern of a frame in the sliding window

        dd = length(temp_vector); % length of unmatched pattern to find if
there are bits deleted in the frame

        dz = round(dd/length(pattern)) * length(pattern) - dd; % determines
the number of deleted bits in the frame

        input3 = [input3 zeros(1,dz) input2(ref_point:end)]; %
Resynchronizes the received code word by adding zeros in front of the frame
in which bits get deleted

        else
            if mod((find1(i) - ref_point),length(pattern)) == 0 ||
find1(i)-find1(i-1) == length(pattern); % Case when the correct frame is
received

            temp_vector = [temp_vector sliding_window((find1(i-
1)):find1(i)-1)]; % stores the frame for resynchronization

            dd = length(temp_vector); % length of frame (unmatched
/matched)

            dz = round(dd/length(pattern)) * length(pattern) - dd; %
determines the number of deleted bits in the frame

            input3 = [input3 zeros(1,dz) input2(ref_point : (find1(i)-1)-
length(pattern))]; % Resynchronizes the received code word by adding zeros
in front of the frame if bits get deleted

            ref_point = find1(i)-length(pattern); % changes reference point
to the current Correctly received frame

            temp_vector = []; % Empty the current temporarily stored frames
            else
                temp_vector = [temp_vector sliding_window((find1(i-
1)):find1(i)-1)]; % Case when if the correct pattern is not found store the
frames
            end
        end
    end
else % Case when one of the synchronization bits get deleted other than the
1st frame of the transmission

    for i = 2:1:length(find1); % Loop through the sliding window to check
for deletions

        if i == length(find1); % case when its the last frame of the
received code word and the synchronization patterns occurs within the
message sequence

            temp_vector = [sliding_window(ref_point:find1(i)-1)]; % to
store the unmatched pattern of a frame in the sliding window

            dd = length(temp_vector); % length of unmatched pattern to find
if there are bits deleted in the frame
```

```
            dz = round(dd/length(pattern)) * length(pattern) - dd; %
determines the number of deleted bits in the frame
            input3 = [input3 zeros(1,dz) input2(ref_point:end)]; %
Resynchronizes the received code word by adding zeros in front of the frame
in which bits get deleted
        else
            if mod((find1(i) - ref_point),length(pattern)) == 0 ||
find1(i)-find1(i-1) == length(pattern); % Case when the correct frame is
recived
                temp_vector = [temp_vector sliding_window(find1(i-
1):find1(i)-1)]; % stores the frame for resynchronization

                dd = length(temp_vector); % length of frame (unmatched
/matched)

                dz = round(dd/length(pattern)) * length(pattern) - dd; %
determines the number of deleted bits in the frame

                input3 = [input3 zeros(1,dz) input2(ref_point:find1(i)-1)];
% Resynchronizes the received code word by adding zeros in front of the
frame if bits get deleted

                ref_point = find1(i); % changes reference point to the
current Correctly received frame

                temp_vector = []; % Empty the current temporarily stored
frames
            else
                    temp_vector = [temp_vector sliding_window(find1(i-
1):find1(i)-1)]; % Case when if the correct pattern is not found store the
frames
            end
        end
    end
end


% decoding

% Input3 is the resynchronized code word that is fed into the decoder

op = vitdec(input3,trellis,tblen,opmode,dectype,puncpat); % Decoder


% Converting Output Back to its corresponding data for each Extended_Prefix
used

out_put = [];

for H = 1:28:length(op)
ham_dist =[];


    for T = 1:1:length(Mapping_Table)
        ham_out = [dec2bin(Mapping_Table(T,2),7) - '0'; op(H:H+6)];
        h_out =  pdist(ham_out,'hamming')*7;
        ham_dist = [ham_dist; h_out];
    end
    indi = find(ismember(ham_dist,min(ham_dist), 'rows'));
    out_put = [out_put, Mapping_Table(indi(1),3),op(H+7:H+27)];
end

x2 = [];
for j2 = 1:m:length(out_put)
    x2 = [x2 bin2dec(sprintf('%d',out_put(j2:j2+(m-1))))];
end
msg2 = gf(x2,m);
msg2 = reshape(msg2,length(msg2)/n,n);
decoded = rsdec(msg2,n,k);
input = reshape(decoded.',1,[]) ;

                    GFInput1 = input;
                    DecOutput2 =[];
                    information_decoded =[];
                    GFRefArray2= gf([0:(2^m)-1],m,prim_poly);
                    for ii=1:length(GFInput1)
                        for k2=0:(2^m)-1
                            temp = isequal(GFInput1(ii),GFRefArray2(k2+1));
                            if (temp==1)
                                DecOutput2(ii) = k2;
                            end
                        end
                    end
                    for u2 = 1:1:length(DecOutput2)
                        information_decoded = [information_decoded
dec2bin(DecOutput2(u2),4) - '0'];
                    end

    BB1 = xor(information1,information_decoded); % Determines the flipped
bits

    BER1 = sum(BB1)/length(information1); % Determines the BER per N
iterations

    BB2 = xor(msg_complete,op); % Determines the flipped bits
    BER2 = sum(BB1)/length(msg_complete); % Determines the BER per N
iterations

data_info = [data_info information1]; % Stores message (mapped information
that was encoded by the encoder) bits for the whole N iterations

op_info = [op_info information_decoded]; % Stores Decoded bits for the
whole N iterations

data_info2 = [data_info2 msg_complete]; % Stores message (mapped
information that was encoded by the encoder) bits for the whole N
iterations

op_info2 = [op_info2 op]; % Stores Decoded bits for the whole N iterations

end
```

```
BB = xor(data_info,op_info); % Determines the flipped bits for the while N
iterations per Probability

BER = sum(BB)/length(data_info) % Determines the BER for the while N
iterations per Probability

BB_inner = xor(data_info2,op_info2); % Determines the flipped bits for the
while N iterations per Probability

BER_inner = sum(BB_inner)/length(data_info2) % Determines the BER for the
while N iterations per Probability

Bit_Error_Rate = [Bit_Error_Rate BER]; % Stores the values of BER for
plotting

Bit_Error_Rate_inner = [Bit_Error_Rate_inner BER_inner]; % Stores the
values of BER for plotting
end

%   plot
close all
figure
plot(probability,Bit_Error_Rate,'b.-');
hold on
plot(probability,Bit_Error_Rate_inner,'mx-');
grid on
legend('outer','inner');
xlabel('probability, Pdel');
ylabel('Bit Error Rate');
title('Bit Error Rate vs Deletion Probability Curve');

end
```