Error Resilient Transmission of H.264 Compressed Video Using CABAC Entropy Coding

David Levine

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science

Concordia University

Montréal, Québec, Canada

September 2007

Your file Votre référence
ISBN: 978-0-494-34705-8
Our file Notre référence
ISBN: 978-0-494-34705-8

ABSTRACT

Error Resilient Transmission of H.264 Compressed Video Using CABAC Entropy Coding

David Levine

This thesis examines the transmission of H.264 compressed video over an Additive White Gaussian Noise (AWGN) channel when Context-based Adaptive Binary Arithmetic Coding (CABAC) is used as the entropy code.

First, several simulations were performed to examine the effect that bit errors have on H.264 decompression. The simulation results show that over 99% of the bit errors tested caused a semantic error to be detected. Further simulation results show that undetected bit errors (bit errors that don't cause semantic errors) were not observed to cause noticeable video quality degradation.

Then, two transmission schemes are proposed. The first proposed scheme uses soft bit information (at the receiver) along with H.264 semantic verification to detect and conceal errors during decompression. Note that this scheme does not involve channel coding. The second proposed scheme is an iterative joint source-channel decoding scheme. It combines a rate-½ convolutional code with the semantic verification strategy of the first scheme. Information is passed from the channel decoder to the source decoder by soft channel decoding. The source decoder then feeds information back to the channel decoder by modifying its original soft values. Simulation results show that both schemes offer significant improvements in terms of subjective quality and in terms of objective quality using PSNR and BER as measures.

Keywords: Video transmission, H.264, CABAC, compression, slice candidates, joint source-channel decoding, semantic verification, soft values, semantic error detection

To my family.

ACKNOWLEDGEMENTS

I would like to begin by thanking Dr. Lynch for proposing an interesting and challenging research topic, for providing continuous support, and for directing my research. I would also like to thank Dr. Le-Ngoc for providing support and guidance. Thanks to all the professors with whom I have interacted throughout my studies at Concordia.

I would like to thank my parents, my brother, my wife and her family for supporting me throughout my studies. I would not have been able to reach my goals without them.

Finally I would like to thank my friends and colleagues for allowing me to share my experience with them. Much of my motivation came from knowing I was not alone.

The Natural Sciences and Engineering Research Council of Canada and the Centre for Advanced Systems and Technologies in Communications supported this research.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AVC	Advanced Video Coding
AWGN	Additive White Gaussian Noise
BD	Blu-Ray Disc
BER	Bit Error Rate
BPSK	Binary Phase Shift Keying
CABAC	Context-based Adaptive Binary Arithmetic Coding
CAVLC	Context Adaptive Variable Length Coding
DBP	Detection Bit Position
DCT	Discrete Cosine Transform
ECSS	Error Concealment using Source Semantics
EDD	Error Detection Delay
EDP	Error Detection Probability
FLC	Fixed Length Code
GOP	Group of Pictures
HD-DVD	High Definition Digital Versatile Disc
HDTV	High Definition Television
ICT	Integer Cosine Transform
IJSCD	Iterative Joint Source-Channel Decoding
JSCD	Joint Source-Channel Decoding
LLR	Logarithm Likelihood Ratio
MB	Macroblock
ML	Maximum Likelihood
MPEG	Moving Picture Experts Group
PSC	Primary Slice Candidate
PSNR	Peak Signal-to-Noise Ratio
RSC	Recursive Systematic Convolutional
SCG	Slice Candidate Generator
SISO	Soft-Input Soft-Output
SNR	Signal-to-Noise Ratio
SOVA	Soft-Output Viterbi Algorithm
SSV	Source Semantic Verifier
VCEG	Video Coding Experts Group
VLC	Variable Length Coding

Chapter 1

Introduction

1.1 Transmission of Compressed Video

Video transmission is a topic that has challenged researchers since its inception. The goal of a video transmission system is to send video information from one location to another. However, transmitting a video sequence requires a large amount of communication resources. Consider a video sequence that runs at 30 frames/sec, has a frame size of 352×240 and uses YUV 4:2:0 color sampling [1]. To transmit this sequence, a bandwidth of about 30 Mbps is needed. Such bandwidth requirements are not practical for most modern communication networks. Thus, compression is used to reduce the required transmission bandwidth. This makes video transmission feasible and efficient.

In addition to transmission, video compression is used in storage. For a given storage device, compression increases the quantity of video data that may be stored. Thus video compression addresses both the issues of transmission and storage.

Several video compression standards have been developed to address the issue of limited bandwidth. The Moving Picture Experts Group (MPEG) [2] has developed several significant video compression formats that have been accepted as worldwide standards, such as MPEG-1, MPEG-2 and MPEG-4 Part 2. These standards have led to a wide range of applications such as the Digital Versatile Disc (DVD), High-Definition Television (HDTV), video-on-demand and mobile video [3],[4].

More recently, there has been a strong demand for higher compression to improve the video quality of current applications. This demand has given rise to a new advanced video compression standard called H.264. This standard refines some of the features of its predecessors while also offering several new ones. H.264 is currently used in a variety of applications including the Blu-ray Disc (BD), High-Definition DVD (HD-DVD), Internet video and videoconferencing [5]. In this thesis, the transmission of H.264 compressed video is examined.

When transmitting video, the communication channel is often noisy. As a result, the received bitstream may contain errors. Several types of transmission errors can occur in a noisy channel including erasures due to dropped packets in network congestion, burst

errors due to multipath propagation in fading channels and random bit errors due to random channel noise [6],[7]. In this thesis, only random bit errors are considered.

While compressed video is smaller than uncompressed video, it is also more susceptible to degradation due to transmission errors. This results from the fact that video compression removes both spatial and temporal redundancy. In uncompressed video, a single bit error will cause only one pixel to be incorrect. On the other hand, a single bit error in compressed video can cause a large portion of video to be corrupted due to error propagation. Errors can propagate both within a frame and between frames.

The effect of error propagation in both senses can be illustrated by an example. Consider the video sequence "Football", compressed using H.264 and transmitted over a noisy channel. As a result of the noise, the compressed sequence has 8 bit errors in frame 1 (and no bit errors in any other compressed frames).

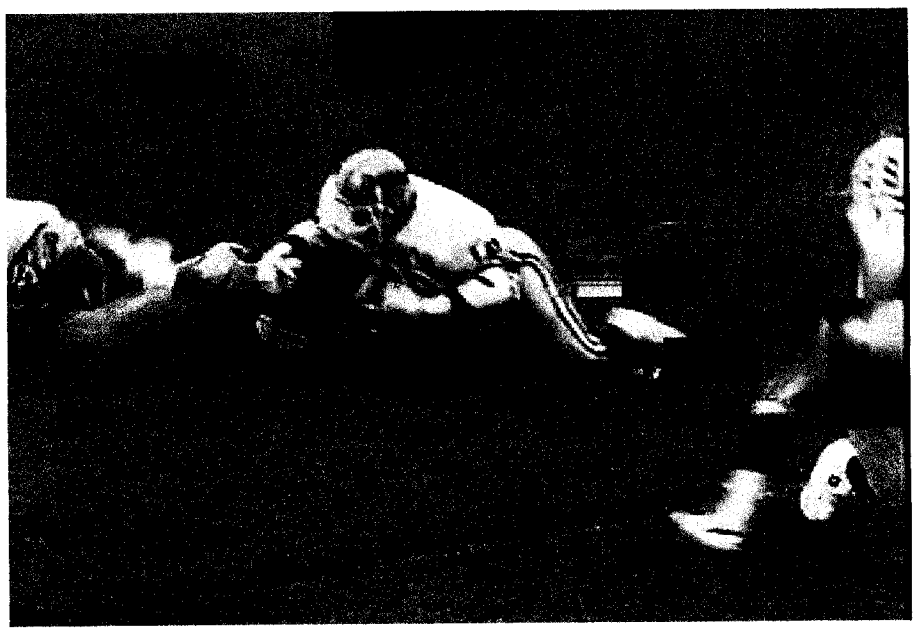

Figure 1.1: The effect of error propagation in the spatial sense is illustrated by Frame 1 of the decompressed video sequence "Football", which contains 8 bit errors in the compressed domain.

Figure 1.1 shows frame 1 after decompression. The figure demonstrates the effect of error propagation in the spatial sense as the frame contains several black boxes and other abnormalities. Figure 1.2 shows frame 3 after decompression. Although none of the bits in the compressed frame are erroneous, frame 3 is predicted using frame 1 as a reference. As a result, the error in frame 1 propagates between frames.

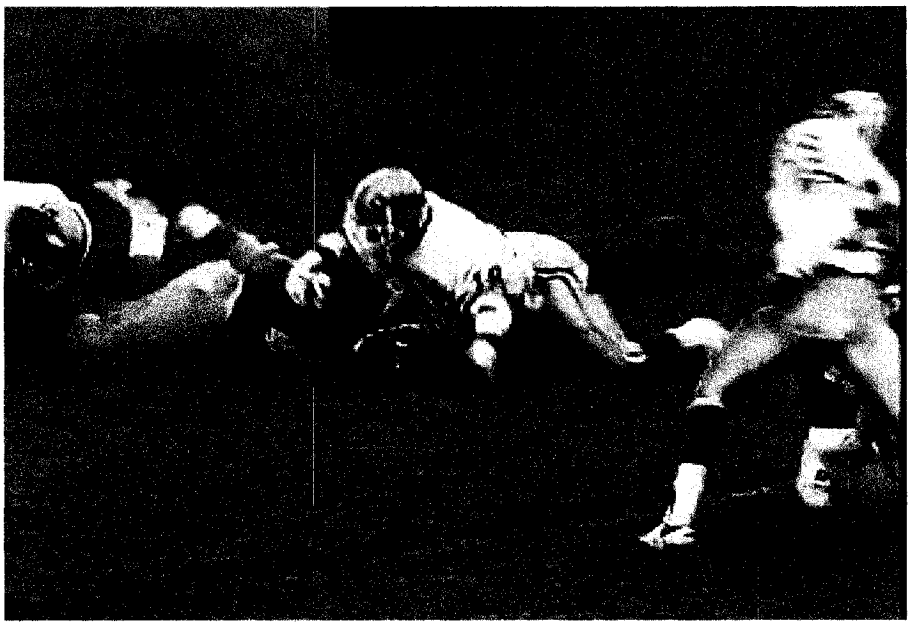

Figure 1.2: The effect of error propagation in the temporal sense is illustrated by Frame 3 of the decompressed video sequence "Football", which contains no bit errors in the compressed domain.

As error propagation may lead to serious quality degradation, measures need to be taken to ensure resilient transmission of compressed video. Two approaches to error resilience are the use of channel coding and the exploitation of residual redundancy in source decoding. Channel coding adds redundant information into a bitstream in order to be able to correct errors [8]. Residual source redundancy can be exploited without any extra information being added into a bitstream [18],[19]. These two approaches can be combined to create a joint source-channel decoding scheme. Several such decoding schemes have been created for various compression standards [18]-[26], [35]-[39]. This thesis examines a transmission scheme that exploits source residual redundancy (without using channel coding) and a transmission scheme using joint source-channel decoding for H.264 compressed video.

1.2 Problem Statement

This thesis investigates methods of improving error resilience in the transmission of H.264 compressed video when Context-based Adaptive Binary Arithmetic Coding (CABAC, described in Section 2.1.3) entropy coding is used. The channel considered in this thesis is an Additive White Gaussian Noise (AWGN) channel and Binary Phase Shift Keying (BPSK) modulation is used for transmission. Two cases of transmission are considered: with channel coding and without channel coding. When channel coding is considered, a rate-½ Recursive Systematic Convolutional (RSC) code is used.

1.3 Figures of Merit

Several figures of merit are used to evaluate the performance of the schemes proposed in this thesis. The figures used are the objective quality using Peak Signal-to-Noise Ratio (PSNR) and Bit Error Rate (BER), the subjective quality, and the complexity.

PSNR is an objective measure of video quality. The PSNR of a video sequence is measured for the luminance (Y), blue chrominance (U) and red chrominance (V) components. The definition of the PSNR for the luminance component of one frame of video is given in Equation 1.1.

$$\mathrm{PSNR}(Y) = \frac{255^2}{\frac{1}{N_Y}\sum_{i=0}^{N_Y}\left(p_{oY}[i] - p_{rY}[i]\right)^2} \tag{1.1}$$

Here, N_Y is the number of pixels in the frame, $p_{oY}[i]$ is the luminance component of a pixel in the original frame and $p_{rY}[i]$ is the luminance component of the same pixel in the reconstructed frame. A higher PSNR indicates that the reconstructed pixels are, on average, closer in value to the original pixels. Note that 8 bits are used to represent each pixel value. Thus, 255 is the maximum possible difference between an original value and a reconstructed value. The equations for the blue and red chrominance PSNR values are essentially the same, except that N_Y, $p_{oY}[i]$ and $p_{rY}[i]$ become N_U, $p_{oU}[i]$ and $p_{rU}[i]$ for blue chrominance and N_V, $p_{oV}[i]$ and $p_{rV}[i]$ for red chrominance.

BER is another objective measure of video quality. The BER is a measure of the number of erroneous bits in a bitstream as a function of the total number of bits, as shown in Equation 1.2. A lower BER indicates a better objective performance.

$$\text{BER} = \frac{\#\ error\ bits}{\#\ total\ bits} \tag{1.2}$$

For subjective video quality, rigorous subjective quality testing is not done due to the considerations of cost and time. Rather, two simple subjective methods are employed. First, for the reader's own assessment, several processed frames are shown along with the originals. Second, the author's comments on subjective quality are presented.

The computational complexity of the schemes proposed in this thesis is also used as a figure of merit. A method that increases performance often does so at the cost of increased complexity. For this reason, it is important to include computational complexity as a figure of merit.

Each of the figures of merit are affected by the amount of the noise in the channel. The channel Signal-to-Noise Ratio (SNR) is used to measure the magnitude of the channel noise. Thus, performance is measured over a range of channel SNR values.

1.4 Thesis Outline

This thesis is organized as follows:

Chapter 2 reviews the background information related to the work in this thesis. The H.264 standard is introduced along with some important coding features of H.264. The H.264 compressor and decompressor used in this thesis are also introduced. The chapter then discusses the transmission of binary information over a noisy channel. Finally, a literature review on joint source-channel decoding of images and video is presented.

Chapter 3 discusses error detection in H.264. First, the chapter describes types of errors that can be detected by the H.264 decompressor. Then, observations are made as to how often bit errors can be detected and how long it takes to detect them. These observations will indicate how useful H.264 source information can be in a joint source-channel decoding scheme.

Chapter 4 presents the first proposed scheme: Error Concealment using Source Semantics (ECSS), which is an H.264 decoding scheme that does not include a channel

code. The results of the proposed scheme are presented objectively, in terms of PSNR and BER, and subjectively. The complexity of the scheme is also presented. While ECSS is not very powerful on its own, it provides a general idea as to how channel information can be used by an H.264 source decoder.

Chapter 5 presents the second proposed scheme: Iterative Joint Source-Channel decoding (IJSCD) of H.264 compressed video. This scheme combines the source decoding of ECSS along with a convolutional channel code. A feedback scheme is introduced where the results of source decoding are iteratively fed back into the channel decoder. The original channel values are modified based on this source information. The objective performance of the proposed scheme is observed in terms of PSNR and BER. The subjective performance is also presented as well as a complexity analysis. The performance of the proposed scheme is compared to the performance of convolutional decoding by itself.

Chapter 6 concludes the thesis and discusses some future work related to error detection and concealment in H.264 as well as some potential improvements to the proposed schemes.

Chapter 2

Background

This chapter presents background information on video compression and decompression using the H.264 standard and transmission of binary information over a noisy channel. Section 2.1 introduces the H.264 video compression standard and discusses the features of H.264 that are relevant to the work in this thesis. Section 2.2 discusses Binary Phase-Shift Keying (BPSK) transmission over an Additive White Gaussian Noise (AWGN) channel. Section 2.3 provides a review of existing schemes for error resilient transmission of video and images. The chapter concludes with a summary in Section 2.4.

2.1 H.264 Video Compression

H.264 is a video coding standard that was developed by the Joint Video Team (JVT), which is a partnership between the Video Coding Experts Group (VCEG) and the Moving Picture Experts Group (MPEG). Each of the two groups has its own version of the standard, however the two standards are jointly maintained so that they always have identical content. VCEG calls its standard H.264/AVC (Advanced Video Coding) while MPEG calls its standard MPEG-4 Part 10 [10].

This section provides background information about H.264 compression relevant to the work in this thesis. Section 2.1.1 provides a general overview of the H.264 standard. Section 2.1.2 briefly examines prediction in H.264. Section 2.1.3 discusses the different entropy coding modes available in H.264. The H.264 compressor and decompressor used in this thesis are described in Sections 2.1.4 and 2.1.5 respectively.

2.1.1 Overview of the H.264 Standard

The H.264 standard [9] was established due to a demand for increased compression in many applications ranging from digital storage to television broadcasting. It is capable of maintaining good video quality at significantly lower bit rates than previous standards such as MPEG-2, and MPEG-4 Part 2, without significantly increasing the design complexity [10].

The H.264 standard comprises a set of profiles [9],[14]. A profile is a set of coding features grouped together to target a specific class of video coding applications. When the standard was first proposed in May, 2003, it contained three profiles: the Baseline Profile, the Main Profile and the Extended Profile [9]. In July, 2004, the standard was amended to

include four additional profiles collectively called the High Profiles [11]. These profiles were designed to support high-resolution video. The differences between the High Profiles are the maximum number of bits per pixel (before compression) and the number of chrominance pixels per frame [11]. Table 2.1 gives a list of the seven current H.264 profiles as well as the applications targeted by each of the profiles.

Profile	**Target Applications**
Baseline	Videoconferencing, mobile video
Main	Broadcast television, video storage
Extended	Streaming video
High	Digital Video Broadcast (DVB), high-end consumer applications (Blu-Ray, HD-DVD), studio distribution
High 10	
High 4:2:2	
High 4:4:4	

Table 2.1: List of H.264 profiles and target applications

Within each profile, compression performance limitations are defined by a set of levels. The level places limits on several parameters including the frame size, the coded bitrate, and memory usage.

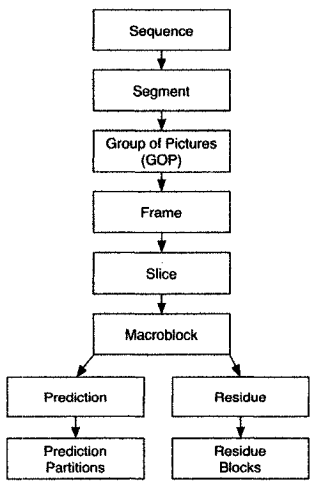

Figure 2.1:Hierarchy of an H.264 compressed bitstream

The general hierarchy of an H.264 compressed bitstream is shown in Figure 2.1. The highest level in H.264 is the sequence, which begins with a Sequence Parameter Set (SPS). The SPS is a header that contains information about parameters common to the entire video sequence such as the profile, level and frame size.

A sequence is comprised of one or more segments, although most sequences are coded with only one segment. A segment begins with a Picture Parameter Set (PPS), which specifies parameters that are common to all frames within the segment such as the entropy coding mode and the initial quantizer parameter.

The next layer below the segment is the Group of Pictures (GOP). A GOP is a series of frames where the number of frames and the type of each frame is specified. In H.264, the GOP is only specified in the encoder parameters. There is no GOP header in the compressed bitstream.

Within each GOP are coded frames. The three main types of coded frames are intra-coded frames (I-frames), predictive frames (P-frames) and bidirectional predicted frames (B-frames) [12]. I-frames are coded using intra prediction (i.e. prediction using pixels in the current frame only). P-frames are coded using both inter prediction (i.e. prediction using pixels in other frames) and intra prediction. B-frames are also coded using inter prediction and intra prediction. Whereas inter prediction in P-frames can only make reference to one other frame, inter prediction in B-frames can make reference to one or two other frames. Like the GOP, there is no frame header in the compressed bitstream.

Each frame is made up of one or more slices. Slices are designed such that there is minimal inter-dependency between them in order to limit error propagation. Each slice begins with a resynchronization point, called a start code, which consists of either a sequence of 23 or 31 zeros followed by a one [9]. A header follows the start code and contains the frame number, the coded frame type, the number of the first macroblock in the slice, and other information required to resynchronize the parameters of the decompressor. The slice header is followed by the slice data. The last byte of slice data is padded with zeros so that the slice is byte aligned. This ensures that the start code of the next slice is also byte aligned.

The slice data is made up of macroblocks, which are 16×16 squares of pixels. Two pieces of information may be coded for each macroblock: a prediction and a residue. The prediction method for all macroblocks in I-frames must be intra prediction. The prediction method for macroblocks in P-frames and B-frames can either be inter prediction or intra prediction [12].

The residue for all macroblocks is subdivided into 4×4 blocks and is transform coded using the Integer Cosine Transform (ICT). The ICT is a modified version of the Discrete Cosine Transform (DCT) designed to allow all operations to be performed using only integer arithmetic and to reduce the total number of multiplications [12].

2.1.2 H.264 Prediction

As mentioned in Section 2.1.1, there are two prediction methods for macroblocks in H.264: inter prediction and intra prediction.

2.1.2.1 Inter Prediction

In H.264, as in previous compression standards, inter predicted macroblocks are predicted from pixels in other frames using block-based motion compensation. However, unlike previous standards, H.264 allows macroblocks to be partitioned for prediction [12]. By doing so, a tradeoff can be made between the number of bits used to represent the prediction and the precision of the prediction. This ultimately improves compression.

There are four ways to partition a macroblock for inter prediction, as seen in Figure 2.2. A macroblock can be partitioned using a single 16×16 partition, using two 16×8 partitions, using two 8×16 partitions or using four 8×8 partitions [12].

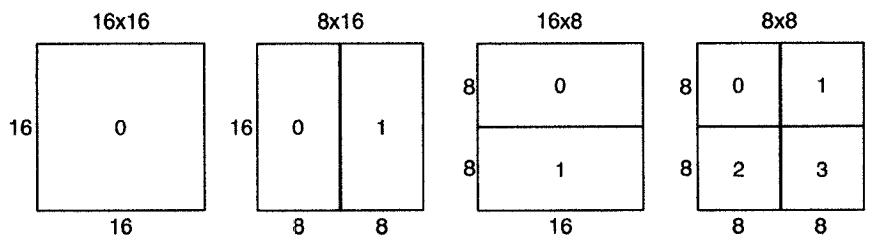

Figure 2.2: Four different way to partition a 16×16 macroblock

When the 8×8 mode is chosen, each partition is sub-partitioned in one of four ways, as depicted in Figure 2.3. A partition can be subdivided as either a single 8×8 sub-partition, two 8×4 sub-partitions, two 4×8 sub-partitions or four 4×4 sub-partitions [12].

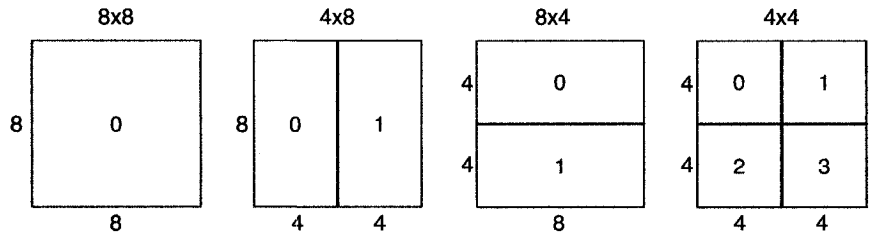

Figure 2.3: Four different way to sub-partition an 8×8 partition

Once the partitioning has been chosen, each partition is predicted from previously encoded frames. Each partition is assigned its own motion vector and the motion vectors are encoded using motion vector prediction [12]. As motion vectors refer to previously encoded pixels, any pixels predicted from erroneous data will also be erroneous. For this reason, errors can propagate between frames (as seen in Section 1.1).

2.1.2.2 Intra Prediction

Intra prediction is used to predict macroblocks without referring to pixels in other frames. To perform intra prediction, each macroblock is partitioned as either 16 4×4 blocks or as one 16×16 block. A prediction mode is then selected for each block [12]. When the 4×4 mode is chosen, there are nine possible prediction modes for each block, as depicted in Figure 2.4. When the 16×16 mode is chosen, there are four possible prediction modes, as depicted in Figure 2.5.

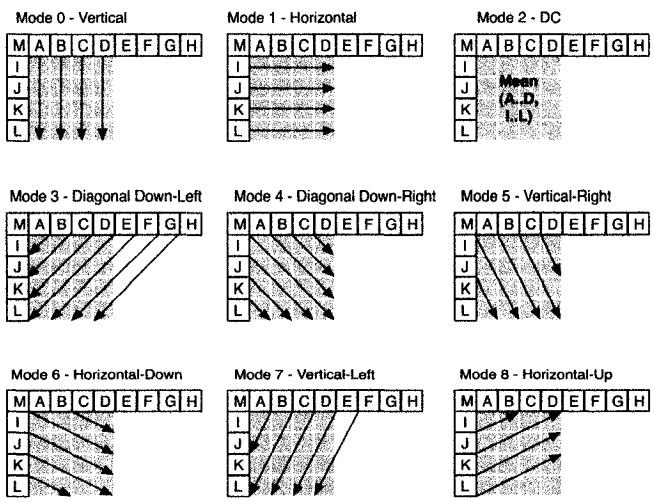

Figure 2.4: The nine possible 4×4 intra prediction modes. Pixels in modes 0 and 1 are extrapolated from a single prediction pixel. Predicted pixels in mode 2 are the average of the pixels directly above to the left of the predicted block. Predicted pixels in modes 3-8 are a weighted average of the relevant pixels [12].

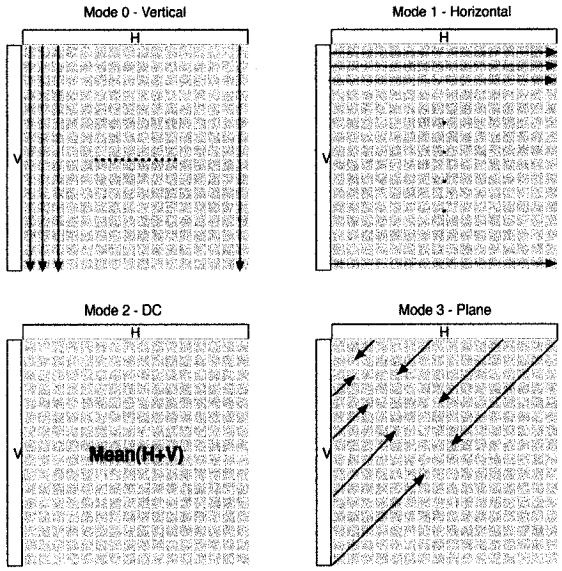

Figure 2.5: The four possible 16×16 intra prediction modes. Pixels predicted in modes 0-2 are determined in the same manner as for 4×4 blocks. Pixels predicted in mode 3 are calculated from the pixels in H and V using a linear plane function [12].

Each prediction mode uses previously encoded pixels above and/or to the left of the predicted block, called prediction pixels. The pixels in the predicted block are called predicted pixels. Each predicted pixel is either an extrapolation of a prediction pixel, an average of the prediction pixels to the left and above the predicted block, or a weighted average of several prediction pixels.

In order for a prediction mode to be usable, the pixels used must be part of the current slice. If the pixels above and to the right of the current block (marked E, F, G and H in Figure 2.4) are not part of the current slice, the rightmost available pixel (marked D in Figure 2.4) is copied into each location and any prediction mode that uses them is considered usable [12]. If there are no pixels available (i.e. the block is at the top-left edge of a slice), no prediction is done and all prediction modes are unusable.

In this thesis, each slice is encoded as one row of macroblocks. Because of this, all blocks at the top edge of each macroblock can't use the prediction pixels above them and blocks at the left edge of the first macroblock can't use the prediction pixels to their left. Thus, there are many blocks in each slice that have unusable prediction modes. While the H.264 compressor always encodes usable intra prediction modes, a bit error can cause the

H.264 decompressor to read the prediction mode incorrectly. As a result, the prediction mode may be unusable. If so, an error is detected. See Section 3.2.1 for details.

2.1.3 Entropy Coding

In all layers above the slice (including the slice header), H.264 syntax elements are entropy coded using either a fixed-length code (FLC) or a variable-length code (VLC) [10]. For syntax elements in the slice layer and below, there are two possible entropy coding modes: VLC mode and Context-based Adaptive Binary Arithmetic Coding (CABAC) mode.

For the work in this thesis, CABAC was chosen as the entropy code for several reasons. First, CABAC achieves a higher coding efficiency than VLC (at a cost of increased complexity) [10]. Second, most research in joint source-channel decoding of compressed video has been done using video compression schemes with VLC entropy coding.

When VLC mode is selected, all syntax elements except for the residual data are coded using zero-order Exp-Golomb codes. For the residual data, a more sophisticated coding method called Context-Adaptive Variable Length Coding (CAVLC) is used to code residual data. In CAVLC, previously coded data is used to switch between VLC tables to take advantage of redundancy in residual data blocks. For more details, consult [9],[10],[12],[15].

When CABAC mode is selected, all syntax elements within the slice layer are coded using CABAC [13]. Figure 2.6 shows the block diagram of the CABAC encoding process. The process consists of three elementary steps: binarization, context modeling and binary arithmetic coding [5],[13],[14].

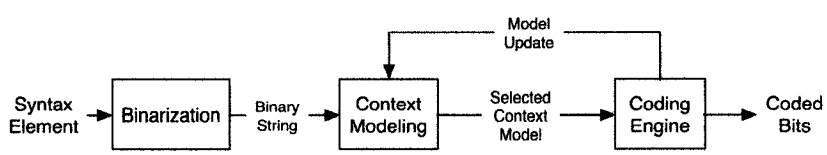

Figure 2.6: Block diagram of CABAC encoding process

In binarization, an H.264 syntax element is mapped onto a unique binary sequence. This simplifies the process of arithmetic coding, as only two possible symbols (zero or one) will be encoded.

In context modeling, a probability model is selected for the binarized syntax element. The probability model depends on the type of syntax element being encoded and may also depend on previously encoded syntax elements.

In binary arithmetic coding, the binarized syntax element is encoded using the selected probability model. As each bit is encoded, the probability model is updated accordingly.

As shall be seen in Section 3.3, when CABAC entropy coding is used, over 99% of bit errors in an H.264 slice can be detected. In Section 3.1, it is discussed that a bit error will cause the CABAC decoder to diverge from the correct decoding path. Due to the arithmetic coding and the adaptive nature of CABAC, it is unlikely for the decoding path to correct itself. Thus, bit errors usually propagate all the way to the end of the slice. This almost always causes a semantic error to be detected, either before reaching the end of the slice or at the end of the slice.

2.1.4 H.264 Compressor

The H.264 standard does not explicitly define the compressor. Rather, the standard defines the syntax of a compressed bitstream and the decoding process. The H.264 compressor [5],[9],[12],[14] used in this thesis is the H.264/AVC reference software version JM 9.6. A block diagram of the compressor is shown in Figure 2.7.

Compression begins by determining the prediction for each macroblock in the frame. As discussed in Section 2.1.2, each macroblock is encoded using either intra prediction or inter prediction. When intra prediction is used, the prediction is formed from pixels in the current slice that have already been encoded, decoded and reconstructed. On the other hand, when inter prediction is used, predicted blocks are formed using motion-compensated samples from one or two references frames stored in either the List 0 or List 1 decoded picture buffer.

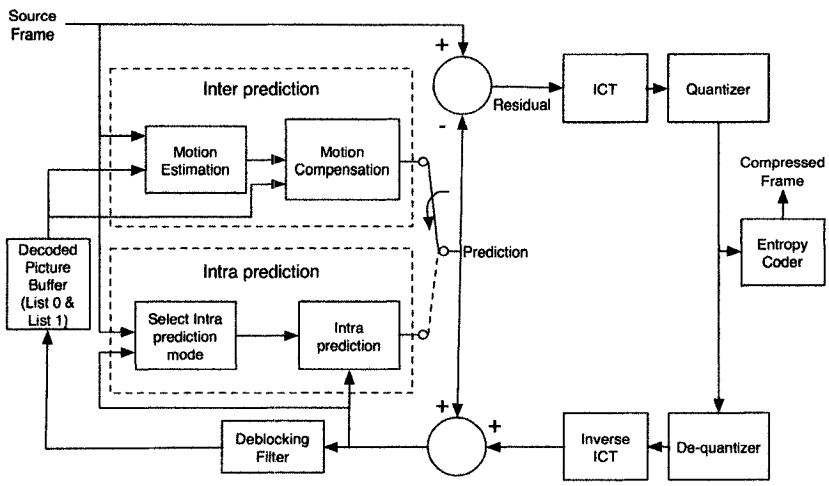

Figure 2.7: H.264 Compressor

Once the prediction method is determined, the residue must be encoded. The residue is determined by subtracting the predicted blocks from the original macroblocks. As discussed in Section 2.1.1, the residue for each macroblock is subdivided into 4×4 blocks. Each 4×4 block is transformed using the ICT and quantized. The quantized values are then entropy coded using one of the prediction modes discussed in Section 2.1.3.

In addition to encoding each macroblock, the compressor also decodes it to produce a reconstructed version of the macroblock. It is these reconstructed macroblocks that are used to carry out both intra prediction and inter prediction. The quantized transform coefficients are de-quantized and inverse transformed to produce a reconstructed residue. Next, the predicted block is added to the reconstructed residue to create the reconstructed macroblock. As shall be seen in Section 2.1.5, the H.264 Decompressor filters the reconstructed macroblocks. Thus, an identical filter is used here. After this process has been applied to all blocks in a frame, the frame is stored in either the List 0 or List 1 decoded picture buffer.

2.1.5 H.264 Decompressor

The H.264 Decompressor [5],[9],[12] used in this thesis, like the compressor, is the H.264/AVC reference software version JM 9.6. However, the decompressor has been slightly modified. A block diagram of the decompressor is shown in Figure 2.8.

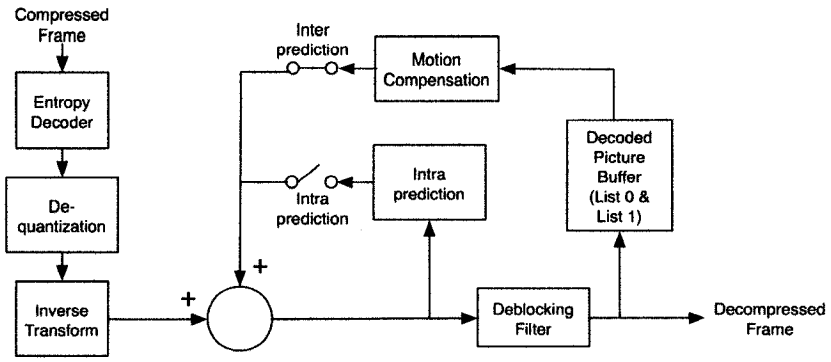

Figure 2.8: H.264 Decompressor

The H.264 Decompressor reconstructs a video sequence from an H.264 compressed bitstream. The H.264 Decompressor first entropy decodes the data for each macroblock to obtain both the prediction information and the quantized transform coefficients. The transform coefficients are de-quantized and inverse transformed as they were in the reconstruction path of the H.264 Compressor. This produces a decompressed residue. The prediction information is identical to the original prediction formed by the H.264 Compressor and is added to the decompressed residue. The resulting blocks are passed through a deblocking filter to reduce visual distortion at block edges. This produces the final decompressed macroblock. The decompressed macroblocks are both displayed and stored in either the List 0 or List 1 decoded picture buffer to be used for future predictions.

As mentioned earlier, the H.264/AVC reference decompressor has been modified. The modified decompressor is able to detect semantic errors that may occur during decompression and decompression does not stop when a semantic error is detected. More details about the detection of semantic errors is given in Section 3.2.

2.2 Binary Transmission over a Noisy Channel

In this thesis, compressed video is transmitted over an Additive White Gaussian Noise (AWGN) channel. When an H.264 slice is transmitted, the receiver must make a decision on each received bit. For a slice of length N, there are 2^N possible decisions. It is desired to associate a probability with each possible decision.

Each bit in the video sequence is modulated using Binary Phase-Shift Keying (BPSK) [16], which maps zeros and ones to -1 and +1 respectively. The modulated values are then transmitted over an AWGN channel, which adds zero mean Gaussian noise to each transmitted value. If u is a bit value that is BPSK modulated and transmitted and y is the received value, then the probability of receiving y given u is shown in Equation 2.1.

$$P(y \mid u) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot \exp\left[-\frac{(y-u)^2}{2\sigma^2}\right] \tag{2.1}$$

The two possible transmitted values, zero and one, are equally probable. Figure 2.9 shows the conditional probability density function of Equation 2.1 for both values of u.

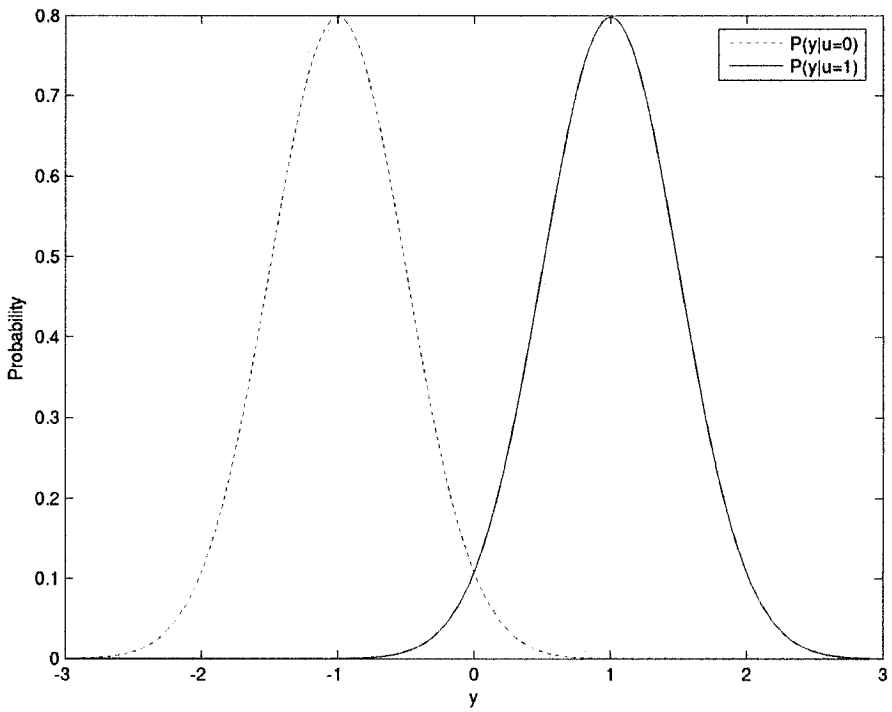

Figure 2.9: Conditional probability density functions for the BPSK receiver

For each received value, the receiver makes a decision (zero or one). The receiver will choose the value that yields the higher probability. Figure 2.9 clearly shows that $y = 0$ is the decision boundary, i.e. the receiver will choose zero if y is less than zero and will choose one if y is greater than zero.

It is desired to determine the probability that the decision made by the receiver is correct. This probability is called the likelihood. If v is the value decided on by the receiver, the likelihood is represented as in Equation 2.2 [16].

$$P(v = u \mid y) = \frac{P(y \mid v = u)}{P(y \mid v = u) + P(y \mid v \neq u)} \tag{2.2}$$

Given the decision boundary, Equation 2.2 can be written as:

$$P(v = u \mid y) = \begin{cases} \dfrac{P(y \mid u = 0)}{P(y \mid u = 0) + P(y \mid u = 1)}, & y < 0 \\ \dfrac{P(y \mid u = 1)}{P(y \mid u = 1) + P(y \mid u = 0)}, & y \geq 0 \end{cases} \tag{2.3}$$

Equation 2.1 can be evaluated for $u = 0$ and $u = 1$. The values can then be substituted into Equation 2.3 and simplified. This results in the likelihood of the receiver's decision v being represented as in Equation 2.4.

$$P(v = u \mid y) = \begin{cases} \dfrac{1}{1 + e^{2y/\sigma^2}}, & y < 0 \\ \dfrac{e^{2y/\sigma^2}}{1 + e^{2y/\sigma^2}}, & y \geq 0 \end{cases} \tag{2.4}$$

When an H.264 slice is transmitted, the likelihood of the received sequence is the product of the likelihood of each decision. In Section 4.2.2, several candidate sequences are determined for each slice (by making the "wrong" decision on certain received values in each candidate). The candidates are then ranked in descending order of likelihood using Equation 2.4.

2.3 Literature Review on Error Resilient Video/Image Transmission

For compressed video and images, error resilient communication often involves exploiting the residual redundancy that remains in the source after compression. This residual redundancy usually appears at the source decoder in the form of syntax/semantic errors. That is, certain bitstreams are syntactically or semantically invalid and cannot be the bitstream that was originally transmitted. This ability of the source decoder to limit the number of possible bitstreams can be used along with soft channel information to

create a decoding scheme that improves error resilience. Several authors have proposed schemes that exploit residual redundancy.

Bystrom *et al.* [19] investigated a scheme for soft decoding of VLC data. The proposed scheme is then used for the transmission of MPEG-4 video. In the scheme, the number of bits and the number of codewords in a coded bitstream are transmitted in an error-free side-channel. When the decoder selects a bitstream from among the candidate bitstreams, the side information is used to limit the number of valid choices. Jeanne *et al.* [20] proposed a similar approach and apply it to the decoding of DCT coefficients in MPEG-4 video. As these schemes deal with soft decoding of VLC codewords and not arithmetic codewords, they could not be applied to an H.264 bitstream encoded using CABAC.

Nguyen and Duhamel [18] proposed a method to decode DCT coefficients in H.263 video taking into account source constraints. In H.263, DCT coefficients are entropy coded using a VLC. Each VLC corresponds to a (*run, level, last*) triplet. The source constraints used in the proposed method are the maximum number of DCT coefficients and the fact that only the last VLC codeword should have a "*last*" of one. This scheme, assumes that the DCT coefficients are the only syntax elements vulnerable to channel errors. All other syntax elements are transmitted in an error-free manner. As was the case for the schemes presented above, this strategy could not be used on an H.264 bitstream that uses CABAC entropy coding.

In addition to exploiting residual redundancy, source decoding can be combined with channel decoding to create joint source-channel decoding (JSCD). In JSCD, the source and channel decoders are designed together with the goal of improving error protection without creating additional bandwidth expansion. JSCD is one of several joint source-channel coding (JSCC) strategies. Other strategies include joint source-channel encoding and joint source-channel rate allocation [17]. Some JSCD schemes involve simultaneously decoding both the source and channel information while others involve passing information from one decoder to the other. In the latter case, information can be fed back and forth between the source and channel decoders in an iterative manner.

Wang and Yu [21] examined a joint source-channel approach to decoding VLC data. In [21], a one-dimensional first order Markov source is VLC encoded, channel coded

using a convolutional code and transmitted over a noisy channel. A MAP decoder was proposed to jointly decode the source and channel codes. The MAP decoder used source and channel statistics to select the most probable source codeword sequence. This strategy was used to decode H.264 motion vectors. In the experiment in [21], VLC entropy coding mode is used (motion vectors are encoded using CAVLC) and all syntax elements other than motion vectors are regarded as error-free. While MAP decoding works for VLC entropy coding mode, it would not work for CABAC mode.

Murad and Fuja [22] observed a high degree of correlation between VLC encoded motion vectors for neighboring macroblocks in MPEG-2 video. This led to the development of a joint source-channel decoding scheme that uses convolutional coding. The scheme creates an integrated graph by combining the graphs for the Markov source model, the Huffman source decoder and the convolutional decoder. The Viterbi Algorithm [41] can then be used on the integrated graph to decode the data. Like in [21], all data other than motion vector information is regarded as error-free.

Pu *et al.* [23] investigated iterative joint source-channel decoding of images. The proposed method uses a Low Density Parity Check (LDPC) code for the transmission of JPEG2000 images over an AWGN channel. In the scheme, the channel decoder performs soft channel decoding and the source decoder performs error detection. After source decoding, the soft channel values are modified based on the error detection results. The modified values are then fed back to the channel decoder to be used in the next iteration. Peng *et al.* [24],[25],[26] proposed a similar scheme using turbo codes for the transmission of JPEG images, MPEG-1 video, vector quantized images, and sub-band coded images. In the schemes in [23],[24],[25],[26], the source decoder only detects errors and does not try to correct errors before feeding information back to the channel decoder.

Another JSCD decoding strategy is to check the syntax and semantics of a received video slice and alter one or more slice bits to try to eliminate any syntax/semantic error. Several authors have employed this strategy.

In Syntax Based Error Concealment (SBEC) [35], MPEG-2 video is channel coded using a simple block-based parity check code. Each coded block consists of 13 bits (12 information bits and 1 parity bit). The coded sequence is transmitted over a noisy

channel. On the receiver side, an error detector checks the parity of each block. If the parity of a block does not add up, the block is flagged as erroneous. If a slice contains one corrupted block, it is decompressed up to 13 times. Each time the slice is decompressed, a different bit in the erroneous block is flipped and the slice syntax is checked. If a slice contains N erroneous blocks, it is decompressed up to 13^N times. Decompression stops as soon as a particular slice configuration does not yield a syntax error. This slice configuration is chosen as the correct bitstream. This scheme usually fails when there is more than one bit error in a 13-bit block.

Syntax and Discontinuity Based Error Concealment (SDBEC) [36] is an extension of SBEC. In contrast to SBEC, SDBEC does not stop after the first error-free decompression. Rather, all slice configurations that do not cause syntax errors are decompressed. A discontinuity measure is used to compare the decompressed slice configurations. The discontinuity measure compares the pixels in the top row of the corrupted slice with the pixels in the bottom row of the slice above it. The slice configuration with the smallest discontinuity measure is chosen as the correct bitstream.

Joint Forward Error Correction / Error Concealment [37] enhances SDBEC by using a stronger channel code for the transmission of MPEG-2 video. The channel code is a block-based (16, 8) quasi-cyclic forward error correction code. A multiple-candidate channel decoder generates a set of slice candidates and groups them by total hamming distance. The hamming distance is a measure of a slice candidate's likelihood. Like in SDBEC, slice candidates are decompressed and checked for syntax/semantic errors. The discontinuity of slice candidates that have no syntax or semantic errors is measured as in SDBEC. A tradeoff is made between the discontinuity measure and the likelihood measure to select the best slice candidate.

Syntax Based Error Concealment using Turbo Codes (SBECTC) [38] combines syntax/semantic checking with turbo decoding to decompress MPEG-4 video. The turbo decoder alleviates the majority of transmission errors. The remaining errors are concealed using source syntax and semantics. For each video slice, the soft-values of the bits at the output of the turbo decoder are used to generate and rank a set of slice candidates in descending order of probability. The slice candidates are checked for syntax and semantic errors. The first slice candidate that does not cause a syntax error is chosen as the winner.

Iterative Joint Source-Channel Decoding using Turbo Codes (IJSCDTC) [39] adds a feedback element to SBECTC. IJSCDTC generates slice candidates and chooses a winning candidate on each turbo decoding iteration (rather than after all iterations, as done in SBECTC). The soft values from the turbo decoder are modified using the winning slice candidate and fed back into the turbo decoder for the next iteration.

The schemes proposed in this thesis build primarily on the work in [38] and [39] in that they use soft values to generate and rank slice candidates. The schemes are presented in Section 4.2 and Section 5.1.

2.4 Summary

This chapter discussed background information pertinent to the research in this thesis. An introduction to H.264 video compression was given. This introduction included an overview of the H.264 standard, and discussed several features of H.264 that are related to the material in the next chapters. The H.264 compressor and H.264 decompressor used in this thesis were also presented. As this thesis deals with transmission of compressed video, a review of binary transmission over a noisy channel was given. Finally, a literature review on joint source-channel decoding of video and images was presented.

Chapter 3

Error Detection in H.264

An overview of the structure and syntax of H.264 and the transmission of binary data over an AWGN channel were discussed in Chapter 2. By transmitting an H.264 bitstream over a noisy channel, bit errors are liable to occur. Chapter 3 examines how bit errors affect the decompression of H.264 video and examines ways for errors to be detected. In this chapter, the effect that bit errors have on H.264 decompression is examined in Section 3.1. Several different types of errors and the ways to detect them are discussed in Section 3.2. Two experiments are then performed. The first experiment examines the error detection probability of H.264 and is discussed in Section 3.3. The second experiment examines the delay in detecting errors and is discussed in Section 3.4. Section 3.5 looks at the effect of undetected bit errors and a summary of the chapter is given in Section 3.6. A conference paper [43] was published based on the contributions of this chapter.

3.1 Effect of Bit Errors on H.264 Decompression

When H.264 video is transmitted over a noisy channel, it is highly susceptible to error propagation. Bit errors affect entropy decoding, which in turn affects the way video is decompressed. This section discusses the effect of bit errors on both entropy decoding and decompressed video quality.

As discussed in Section 2.1.3, there are 2 choices of entropy codes in H.264: VLC and CABAC. The entropy code used in this thesis is CABAC, which is an arithmetic code. If a single bit in a CABAC stream is erroneous, then its encompassing CABAC syntax element will be erroneous. In CABAC, the decoding of a syntax element is dependent on the decoding of previous syntax elements. Thus, an erroneous syntax element could cause the CABAC decoder to diverge from the correct decoding path.

CABAC is also a context-based and adaptive entropy code. The set of possible syntax elements and the way the arithmetic code bits map to syntax elements change as syntax elements are decoded [13]. Thus, even if the decoding path converges with the correct decoding path, future syntax elements could still be incorrect. They would only be correct if both the possible syntax elements and the context are correct. In short, since CABAC is both an arithmetic code and is context-based adaptive, bit errors will often propagate in an H.264 video sequence when CABAC is used.

When an error propagates in an H.264 sequence, the decompressed video becomes corrupted. The corruption continues until the decompressor reaches a resynchronization point. As discussed in Section 2.1.1, the resynchronization points in H.264 are the start codes that prefix each slice header. Thus, within a frame, the video corruption caused by an error never propagates beyond the end of a slice.

Although resynchronization points prevent errors from propagating beyond the end of a slice, error propagation still occurs between frames. As discussed in Section 2.1.1, P-frames and B-frames use inter prediction. Inter prediction uses motion vectors to refer to pixels in one or two reference frames. Even if a motion vector is correct, the predicted pixels could still be corrupted if the decompressor generates them using corrupted pixels in one of the reference frames.

3.2 Detecting Errors in H.264

In many cases where a bit error corrupts a video sequence, the H.264 decompressor is capable of detecting that an error occurred. For compressed data in general, there are two kinds of detectable errors: syntax errors and semantic errors. Syntax errors occur when entropy decoded syntax elements are invalid. Semantic errors occur when the decompressor is instructed to perform a task known to be incorrect.

All syntax elements in CABAC are valid, i.e. there are no CABAC syntax errors. Hence, with CABAC as the entropy code, all errors that can be detected by the H.264 decompressor are semantic errors.

Semantic errors are caused by bit errors. Several types of semantic errors are detected in the same slice as the bit errors that cause them. If one of these types of semantic errors is detected, the slice in which it is detected has at least one bit error in it. The bit at which the semantic error is detected is called the detection bit. The bit error or bit errors that cause the semantic error must be located between the first bit in the slice and the detection bit.

When a semantic error is detected, the decompressor stops checking for more errors since only one semantic error is needed to know that a slice contains bit errors. Thus every slice has at most one semantic error.

There are several types of H.264 semantic errors that can be detected in decompression. Among the most common of these error types are intra prediction mode errors, slice fragment errors, slice run-on errors, macroblock-overrun errors and illegal reference index errors. This section gives a detailed description of these types of semantic errors and the events that cause them to occur.

3.2.1 Intra prediction Mode Errors

As mentioned in Section 2.1.2, intra prediction estimates a block (or macroblock) using neighboring pixels. The pixels used for prediction must always be from the same slice as the block being predicted.

All intra prediction modes are available to predict any block. However, for blocks on the top edge and/or the left edge of a slice, some of the intra prediction modes are unusable. For instance, vertical prediction is unusable on a block at the top edge of a slice since the pixels above the block are not from the same slice. When an intra prediction mode is unusable, it is called an invalid intra prediction mode. Modes that can be used are called valid intra prediction modes. The H.264 compressor always uses valid intra prediction modes. Therefore, when the H.264 decompressor reads an invalid intra prediction mode, it detects an intra prediction mode error.

3.2.2 Slice Fragment Errors

In H.264 decompression, there is an end-of-slice flag that tells the decompressor that the current slice is finished. When the end-of-slice flag is read, any unread bits in the slice are ignored. In most cases, the only bits in an H.264 bitstream that should be ignored are the bits used to pad the last byte in a slice. Therefore, the decompressor should read the end-of-slice flag in the last byte of the slice. If the H.264 decompressor reads the end-of-slice flag before it has reached the last byte of a slice, it detects a slice fragment error.

3.2.3 Slice Run-On Errors

In contrast to slice fragment errors, slice run-on errors occur when the decompressor reaches the start code for the next slice without ever reading the end-of-slice flag for the

current slice. Since the decompressor must read the end-of-slice flag to move on to the next slice, it detects a slice run-on error.

Slice run-on errors only occur when the decompressor has reached the end of the slice data. Thus, the detection bit of a slice run-on error is always the last bit of the slice.

3.2.4 Macroblock-Overrun Errors

In a slice, the H.264 decompressor will continue to decode macroblocks as long as it doesn't read the end-of-slice flag and as long as a slice run-on error doesn't occur. However, the number of macroblocks it decodes may not be correct.

As mentioned in Section 2.1.1, H.264 slice headers contain the macroblock number of the first macroblock in the slice. When decompressing a slice, the decompressor scans ahead to the next slice and reads its header to determine the macroblock number of the first macroblock. Using this macroblock number, the decompressor can determine the correct number of macroblocks for the current slice. If the decompressor reads the end-of-slice flag with too few macroblocks decoded, or if it doesn't read the end-of-slice flag with the correct number of macroblocks decoded, it detects a macroblock-overrun error.

3.2.5 Illegal Reference Index Errors

Illegal reference index errors occur in inter-predicted macroblocks or blocks. In an inter-predicted frame, blocks are predicted using pixels in other frames. The frames that are used for prediction are stored in two reference lists. The number of reference frames that can be placed in the reference lists is fixed. However, the reference lists are not always full. If the decompressor tries to predict a block using an index to an empty element in one of the reference lists, it detects an illegal reference index error.

3.3 Estimating the Error Detection Probability of H.264

Section 3.2 described the types of semantic errors that can be detected by the H.264 decompressor. This section discusses the Error Detection Probability (EDP), which is a measure of how often bit errors cause semantic errors.

Section 3.2 discussed that a slice that has a semantic error must also have at least one bit error. However, the inverse is not always true. That is, a slice that has one or more bit

errors does not always have a semantic error. Rather, a slice that has bit errors has a certain probability of having a semantic error. This probability is the EDP.

To estimate the EDP, a simulation was performed. This section describes how the simulation was performed, shows the results and discusses the implications of the results.

3.3.1 Description of Simulation

The EDP simulation consisted of inserting bit errors into a compressed video sequence and observing whether or not these errors caused the decompressor to detect a semantic error.

To keep the simulation from becoming too complex, only one bit error was inserted in the video sequence for each run. One simulation run was performed for each slice data bit in the entire sequence. No runs consisted of inserting errors in sequence headers, picture headers or slices headers.

3.3.2 Simulation Results

3.3.2.1 Results for Video "Football"

The 4-second video sequence "Football" has a frame size of 352x240, and runs at 30 frames per second. The sequence is compressed in H.264 at 1 Mb/s using the Main Profile and CABAC entropy coding. The sequence uses a group of pictures (GOP) of length 15 with IBBPBBP structure.

Error Type	I-Slices		P-Slices		B-Slices		Total	
	# Runs	Prob.	# Runs	Prob.	# Runs	Prob.	# Runs	Prob.
Intra pred. Mode	819305	0.929	518701	0.353	77283	0.046	1415289	0.351
Slice Run-On	23812	0.027	173390	0.118	210005	0.125	407207	0.101
Slice Fragment	7055	0.008	70532	0.048	100803	0.060	178390	0.044
MB Overrun	25576	0.029	697970	0.475	1281872	0.763	2005418	0.497
Undetected	6173	0.007	8816	0.006	10080	0.006	25069	0.007
Total	881921	1.000	1469409	1.000	1680043	1.000	4031373	1.000

Table 3.1: Probability of detecting each error type and probability of undetected bit errors for video sequence "Football" when each simulation run contains one erroneous slice data bit

Table 3.1 shows the probability of each error type for this sequence as well as the frequency of the undetected bit errors. The results are shown for all bits in I-slices, all bits

in P-slices and all bits in B-slices. The results are also shown for all bits in the entire sequence.

It is observed that the majority of errors detected in I-slices are intra prediction mode errors. The majority of errors detected in P-slices are macroblock-overrun errors, although there are a substantial number of intra prediction mode errors in the P-slices as well. The B-slices are dominated by macroblock-overrun errors while very few intra prediction mode errors occur in the B-slices. Overall, less than 1% of bit errors in any slice type go by undetected, which corresponds to an EDP of over 0.99. The EDP is, however, slightly lower in the I-slices than it is in the P-slices and the B-slices, as the probability of undetected bit errors is higher.

It was hypothesized that bit errors close to the end of a slice have a lower EDP than bit errors far from the end of a slice. It was believed that it takes, on average, a certain amount of time for a bit error to lead to a semantic error. The closer a bit error is to the end of a slice, the less of an opportunity it should have to produce a semantic error. To test this hypothesis, the probability of undetected bit errors was estimated as a function of distance from the end of the slice. This probability corresponds to 1 minus the EDP.

Figure 3.1 shows the probability of undetected bit errors as a function of the distance between those bit errors and the end of the slice. Each data point on the curve is the center of a bin that is 25 bits wide.

As mentioned in Section 2.1.1, H.264 slices are byte aligned. As such, there is a possibility that the last few bits in the last byte are padding bits and have no effect on decompression. If so, then any bit error that occurs in one of those bits will never be detectable (and will have no deleterious effect on the decompressed video). Thus, the data for bits less than 7 bits away from the end of the slice are discarded. The center of the first bin is therefore 19 bits from the end of the slice.

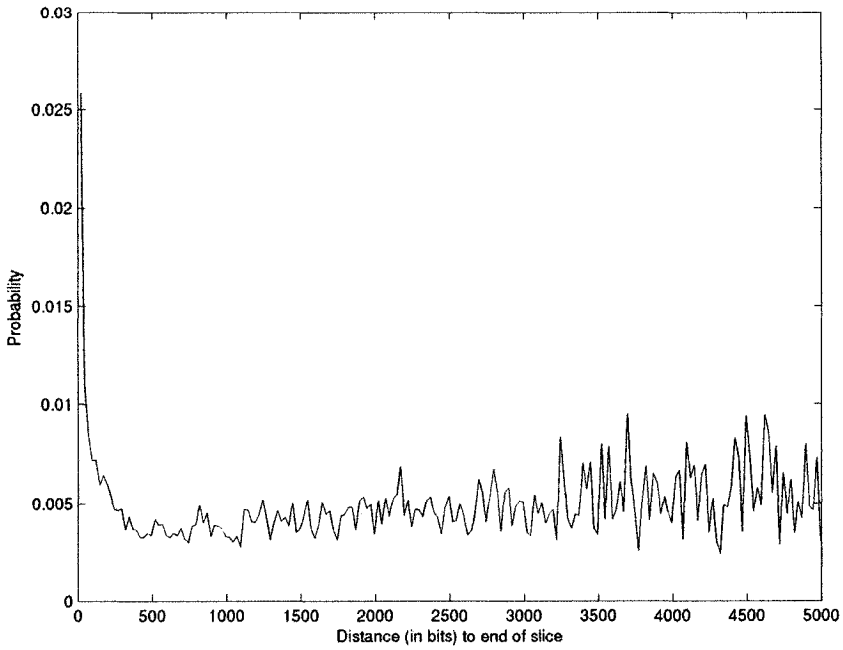

Figure 3.1: Probability of undetected bit errors as a function of distance to the end of the slice for video "Football"

It is noted that the curve in Figure 3.1 becomes noisier as the data points move farther to the right. This is due to the fact that not all video slices are the same length. Any Slice shorter than the lower bound of a bin contributes no data to that bin. When moving to the right, the number of slices that contribute data decreases. As a result, there is more noise in the curve.

The results show that the probability of undetected bit errors does in fact decrease as a function of the distance from the end of the slice (and thus, the EDP increases). However, at a certain point, the probability stops decreasing and begins to increase slightly. The reason for this can be observed when the results are broken down by slice type. Figure 3.2 shows the probability versus distance curve for only the I-slices. Figure 3.3 shows the same curve for only the P-slices and Figure 3.4 shows the curve for only the B-slices.

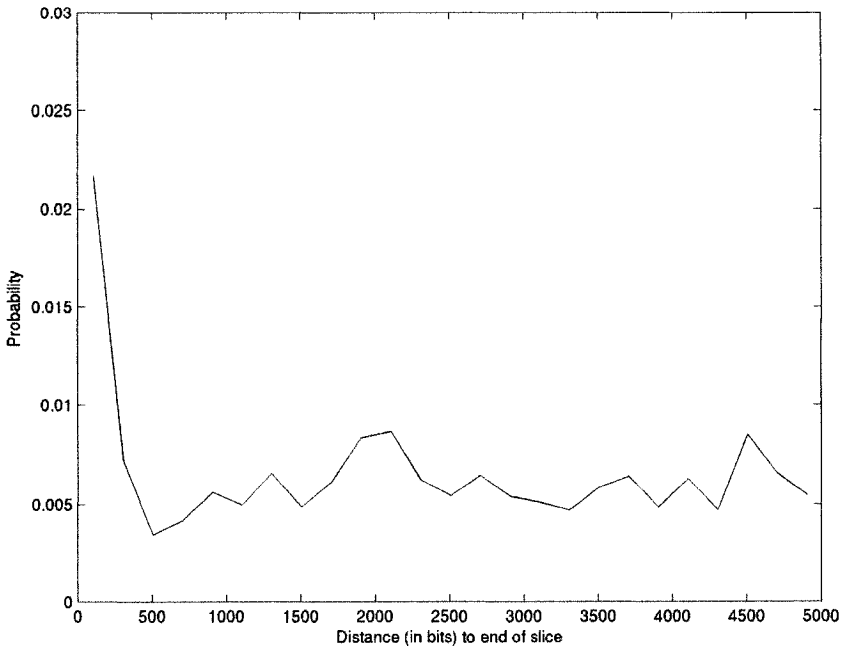

Figure 3.2: Probability of undetected bit errors as a function of distance to the end of the slice for I-slices of video "Football"

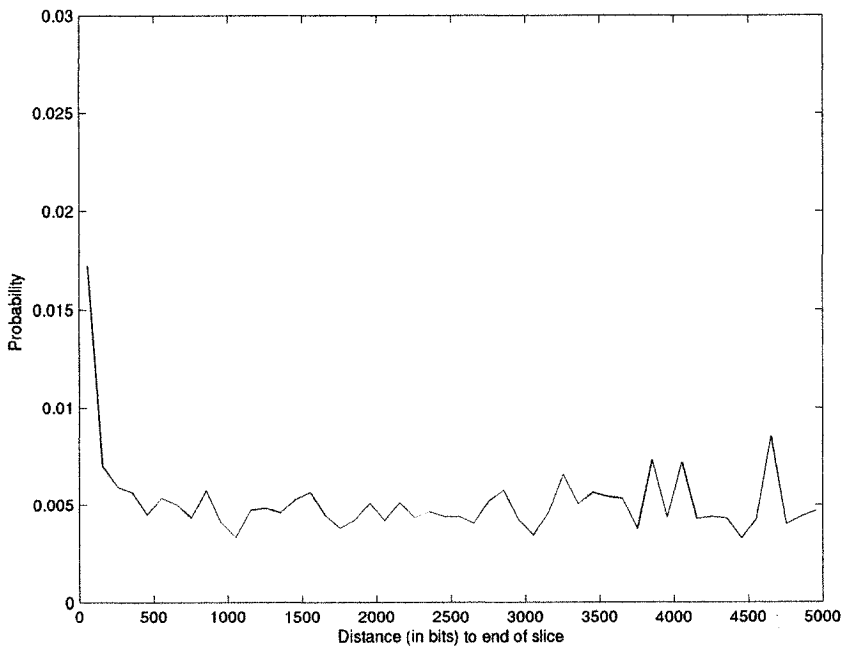

Figure 3.3: Probability of undetected bit errors as a function of distance to the end of the slice for P-slices of video "Football"

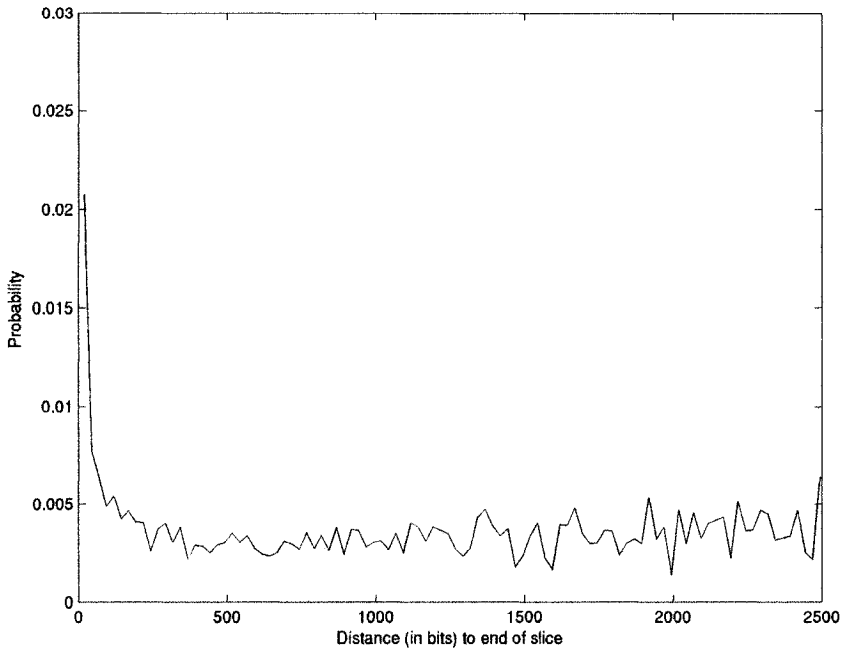

Figure 3.4: Probability of undetected bit errors as a function of distance to the end of the slice for B-slices of video "Football"

To ensure that each bin has a sufficient amount of data, the bins for the curve for the I-slices are 100 bits wide, the bins for the curve for the P-slices are 50 bits wide and the bins for the curve for the B-slices are 25 bits wide. Also, the curve for the B-slices is only shown up to a distance of 2500 bits from the end of the slice as compared to 5000 bits for the other curves. The reason for this is that B-slices are, on average, much shorter than I-slices or P-slices and there are not enough B-slices longer than 2500 bits to get useful results.

Comparing the 3 curves, it is noticed that they all exhibit the same initial behavior: the probability of undetected bit errors decreases as a function of distance. In all three curves, the probability stops decreasing at a certain point and remains relatively flat. However, the flat section of the curve for the I-slices is slightly higher than the flat section of the curve for the P-slices and the B-slices. Based on the average EDP results from Table 3.1, this makes sense since the probability of undetected errors for I-slices was found to be slightly higher than for P-slices and B-slices.

As the distance to the end of the slice increases, the percentage of data points coming from I-slices increases and the percentage of data points coming from P-slices and B-

slices decreases. This is the reason why the curve in Figure 3.1 begins to slightly increase.

3.3.2.2 Results for Video "Table-Tennis"

The video sequence "Table-Tennis" is compressed under the same conditions as the video sequence "Football". Like "Football, it is a 4-second sequence with a frame size of 352x240, and runs at 30 frames per second. Table 3.2 shows the probability of each error type as well as for the undetected bit errors for all bits in I-slices, P-slices B-slices and for all slice types in the sequence.

Error Type	I-Slices		P-Slices		B-Slices		Total	
	# Runs	Prob.	# Runs	Prob.	# Runs	Prob.	# Runs	Prob.
Intrapred. Mode	1027355	0.915	399947	0.249	64219	0.064	1491521	0.400
Slice Run-On	29193	0.026	301968	0.188	314072	0.313	645233	0.173
Slice Fragment	7859	0.007	72280	0.045	48166	0.048	128305	0.034
MB Overrun	51648	0.046	823988	0.513	567940	0.566	1443576	0.387
Undetected	6737	0.006	8031	0.005	9031	0.009	23799	0.006
Total	1122792	1.000	1606214	1.000	1003428	1.000	3732434	1.000

Table 3.2: Probability of detecting each error type and probability of undetected bit errors for video sequence "Table-Tennis" when each simulation run contains one erroneous slice data bit

The same general trends that were seen for "Football" can be seen for "Table-Tennis". The intra prediction mode errors dominate the I-slices while the macroblock-overrun errors are the most frequent errors in both the P-slices and the B-slices. One noticeable difference between "Football" and "Table-Tennis" is the fact that the average probability of undetected bit errors for the B-slices is higher for "Table-Tennis".

Figure 3.5 shows the probability of undetected bit errors as a function of the distance between those bit errors and the end of the slice for "Table-Tennis". This curve follows the same general trend as in "Football".

Figure 3.6 shows the probability of undetected bit errors versus distance curve for only the I-slices of "Table-Tennis". Figure 3.7 shows the curve for only the P-slices and Figure 3.8 shows the curve for only the B-slices. These curves are plotted using the same parameters as the curves for "Football". As in "Football", the curve for the I-slices flattens out at a higher level than the curves for the P-slices and the B-slices.

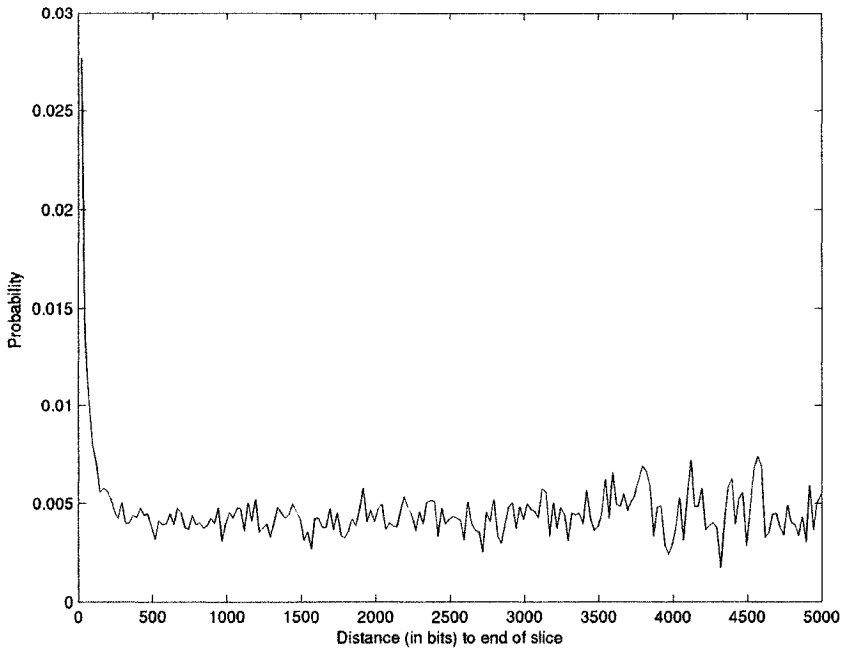

Figure 3.5: Probability of undetected bit errors as a function of distance to the end of the slice for video "Table-Tennis"

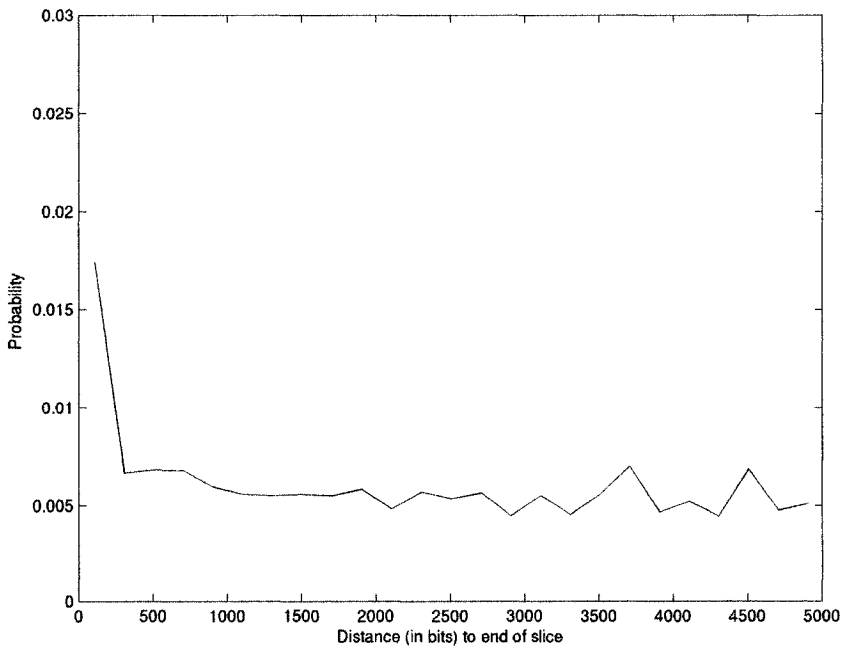

Figure 3.6: Probability of undetected bit errors as a function of distance to the end of the slice for I-slices of video "Table-Tennis"

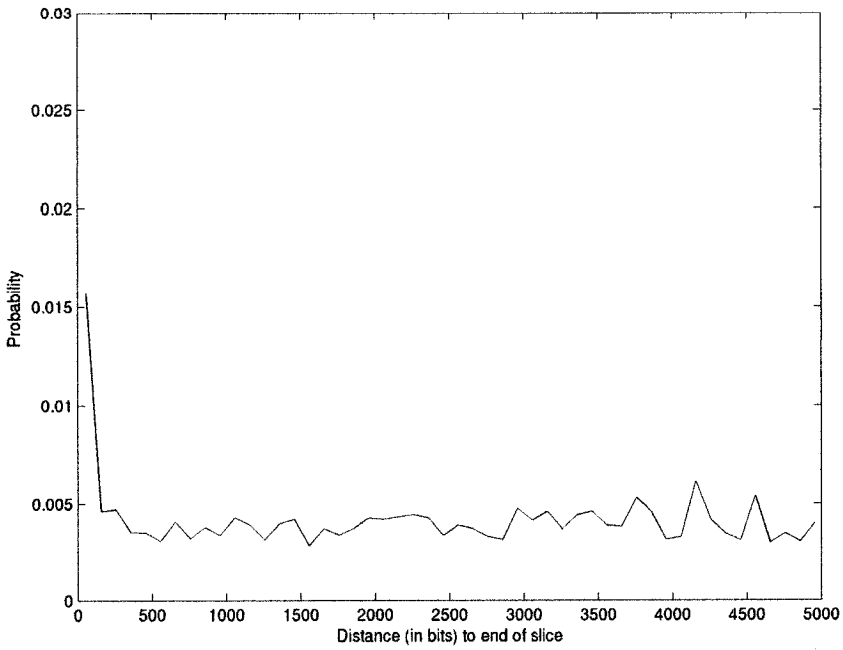

Figure 3.7: Probability of undetected bit errors as a function of distance to the end of the slice for P-slices of video "Table-Tennis"

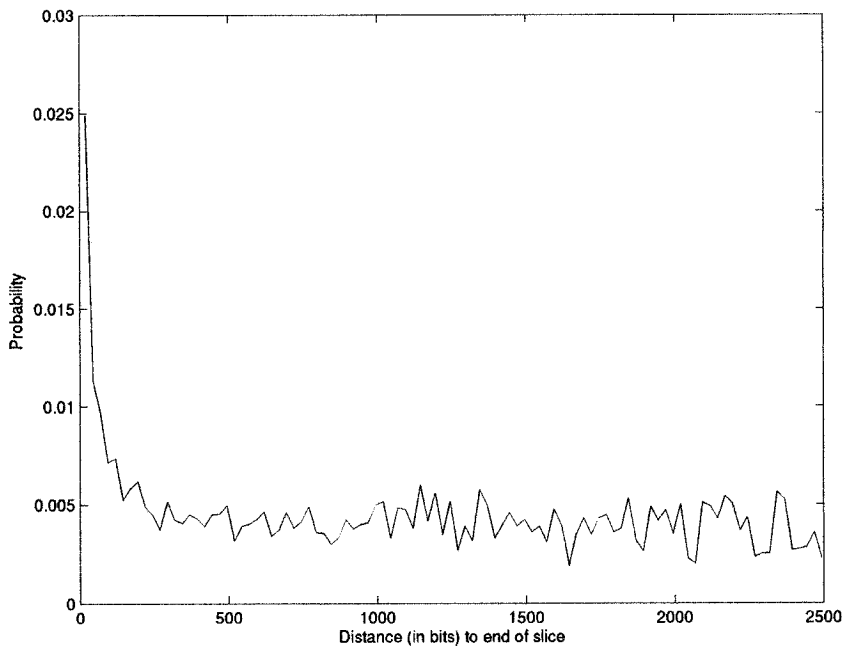

Figure 3.8: Probability of undetected bit errors as a function of distance to the end of the slice for B-slices of video "Table-Tennis"

3.3.3 Implications of the Simulation Results

It was observed that bit errors, on average, cause semantic errors more than 99% of the time. This implies that a slice has a high probability of not having any bit errors if the decompressor doesn't detect any semantic errors.

It was also observed that bit errors farther from the end of a slice are more likely to cause semantic errors than bit errors closer to the end of a slice. Thus, if the decompressor doesn't detect a semantic error, there is more certainty in the correctness of bits farther away from the end of a slice than of bits closer to the end of a slice.

3.4 Estimating the Error Detection Delay in H.264

In Section 3.3, observations were made as to how often slices with bit errors have semantic errors. In this section, the focus is on the cases where semantic errors are detected. In each case, the Error Detection Delay (EDD) is observed.

The EDD is a random variable that represents the distance, in bits, between a bit error and its resulting detection bit. It is desired to estimate the probability density function of the EDD.

The probability density function of the EDD was estimated through simulation. This section describes how the simulation was performed, shows the results and discusses the implications of the results.

3.4.1 Description of Simulation

The EDD simulation, like the EDP simulation, consisted of inserting errors into in a compressed video sequence and performing many simulation runs. Once again, only one bit error was inserted per run and runs were done for every bit in the sequence except the sequence, picture and slice headers. In each run, the value of the observed EDD was recorded. If no semantic error was detected, the run was disregarded.

Let B be a random variable denoting the location of a bit error in a slice. It was assumed that the EDD and B are independent. In other words, the location of a bit error does not affect the EDD (and vice versa).

Let D_n be a particular EDD value. Without assuming the EDD and B are independent, $P(D_n)$ depends on B. With the EDD and B independent, $P(D_n)$ can be estimated as the number of runs in which D_n is observed divided by the total number of runs.

3.4.2 Simulation Results

3.4.2.1 Results for Video "Football"

Figure 3.9 shows the EDD probability histogram for the video "Football". Figure 3.10 shows the part of the histogram up to a delay of 300 bits. It is observed from Figure 3.10 that the histogram reaches an initial peak at 11 bits and drops down dramatically. It then increases until it reaches a second peak at an EDD of approximately 70 bits. After reaching this peak, Figure 3.9 shows that the curve drops down and eventually levels off as the probability begins to approach zero near a bit delay of 1000 bits.

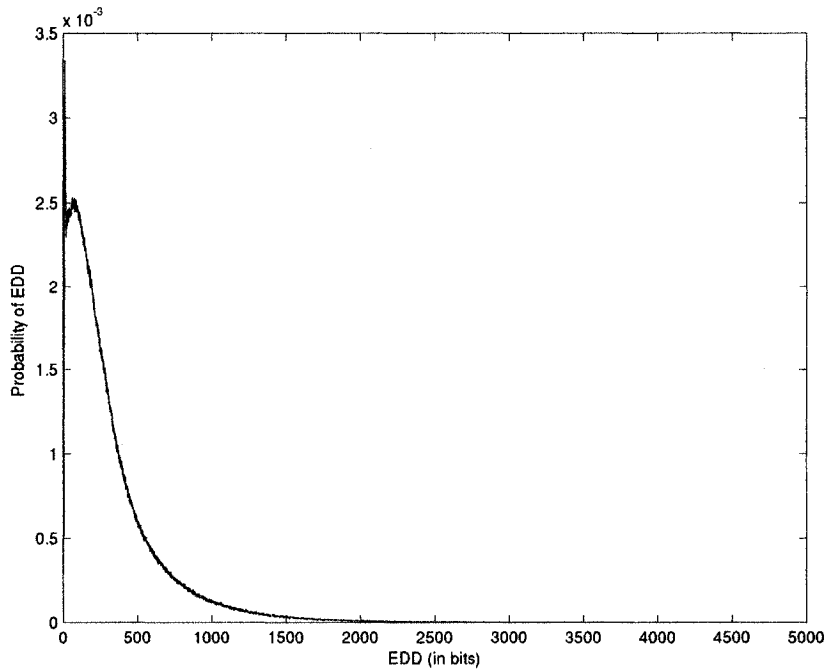

Figure 3.9: Histogram of EDD probability for video sequence "Football". In each run, a semantic error is caused by a single bit error in a known location.

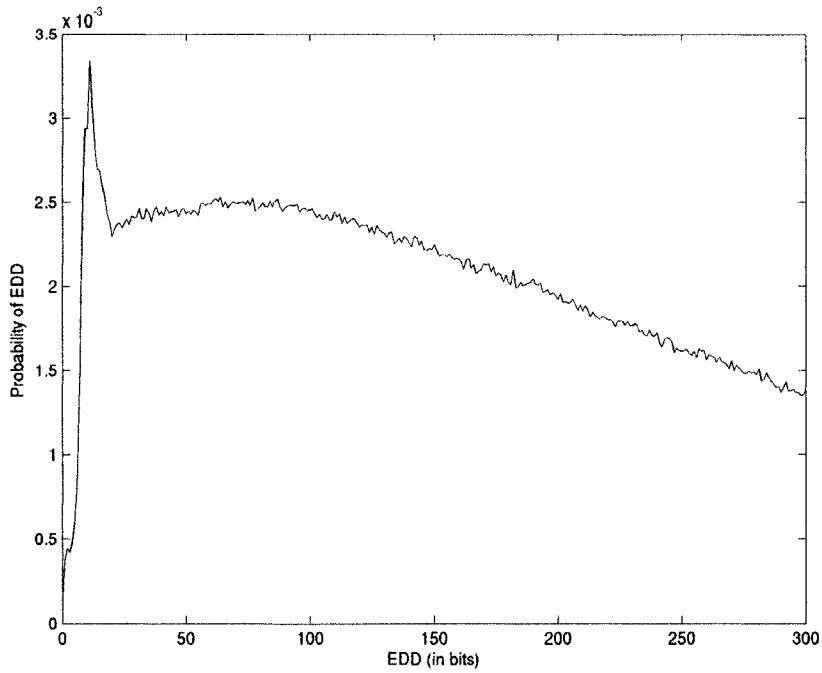

Figure 3.10: Segment of EDD probability histogram for video sequence "Football" up to a delay of 300 bits

These results indicate that bits errors that take a long time to lead to semantic errors are rare and bit errors that lead to semantic errors quickly are more common. However, the probabilities at the peaks are very low. The peak at 11 bits has a probability of 3.4×10^{-3} and the peak at 70 bits has a probability of 2.5×10^{-3}. This means that, though small EDD values are more common than large EDD values, there are still a significant number of large EDD values.

Figure 3.11 shows the cumulative EDD probability. That is, for each data point on the curve, the probability is the percentage of runs for which the EDD is less than or equal to a particular distance.

Since this measure is a cumulative probability estimate, the curve is a monotonically increasing curve. It can be observed that the curve has a knee point at around 750 bits. Also, just over 90% of the runs result in EDD values less than 750 bits.

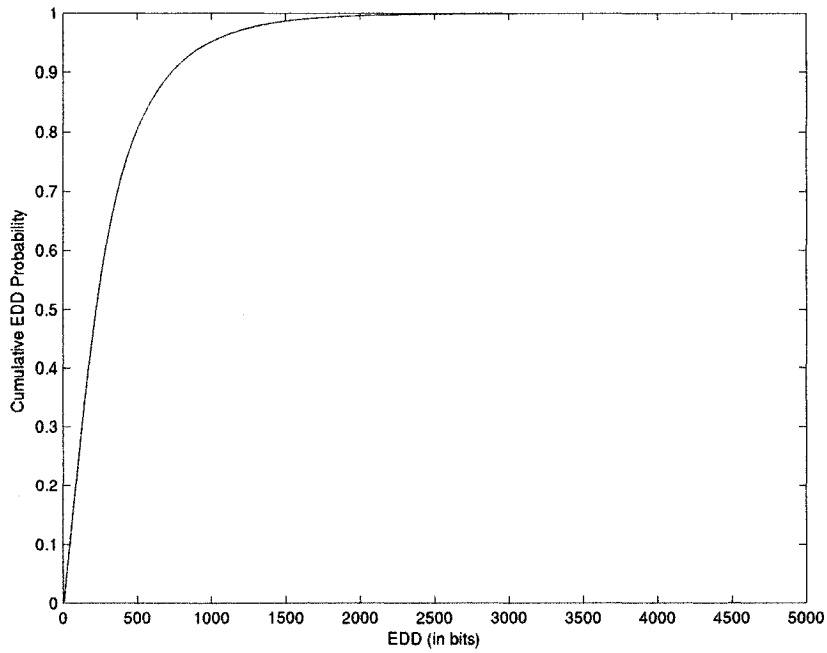

Figure 3.11: Histogram of cumulative EDD probability for video "Football"

Figure 3.12, Figure 3.13, and Figure 3.14 show the EDD histograms for the I-slices, the P-slices and the B-slices respectively. All three of the slice types exhibit similar characteristics in their delay curves, with some variations. For instance, the sharp drop at the beginning of the curve for the P-slices is significantly lower than the initial drops for the I-slices or the B-slices. The tail of the P-slice curve also lasts slightly longer than the tails for the other two slice types. In general, however, once the peak value is reached, the EDD probability decreases as the delay increases.

Figure 3.15, Figure 3.16, and Figure 3.17 show the cumulative EDD probability for the I-slices, the P-slices and the B-slices respectively. All three curves increase monotonically. The curves for the I-slices and the B-slices appear to have similar knee points near 550 bits. The knee point on the curve for the P-slices, on the other hand, is farther to the right at approximately 1200 bits. This implies that for this sequence, it takes more time on average to detect errors in P-slices than in either I-slices or in B-slices. In all three cases, the probability at the knee point is slightly above 0.9.

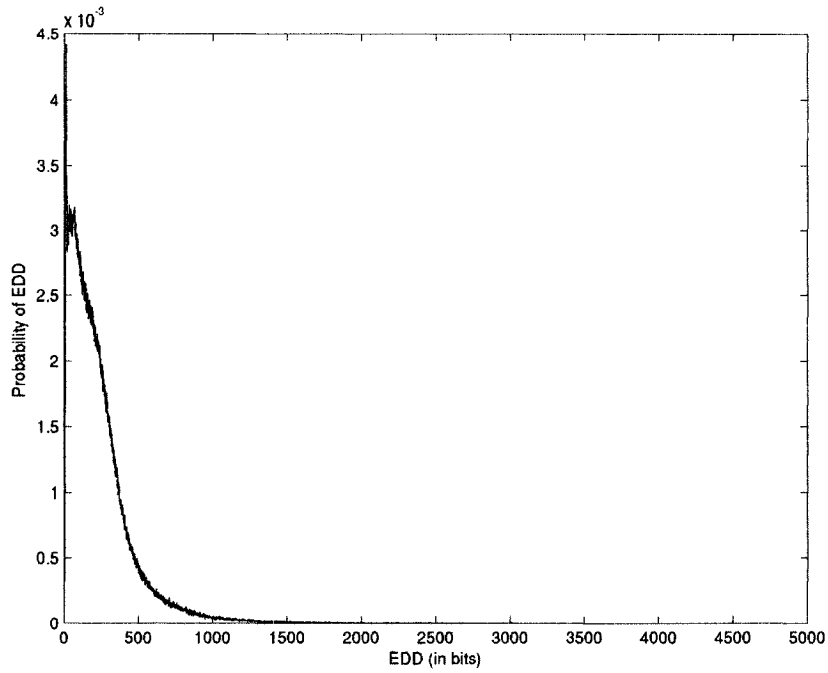

Figure 3.12: Histogram of EDD probability for the I-slices of video "Football"

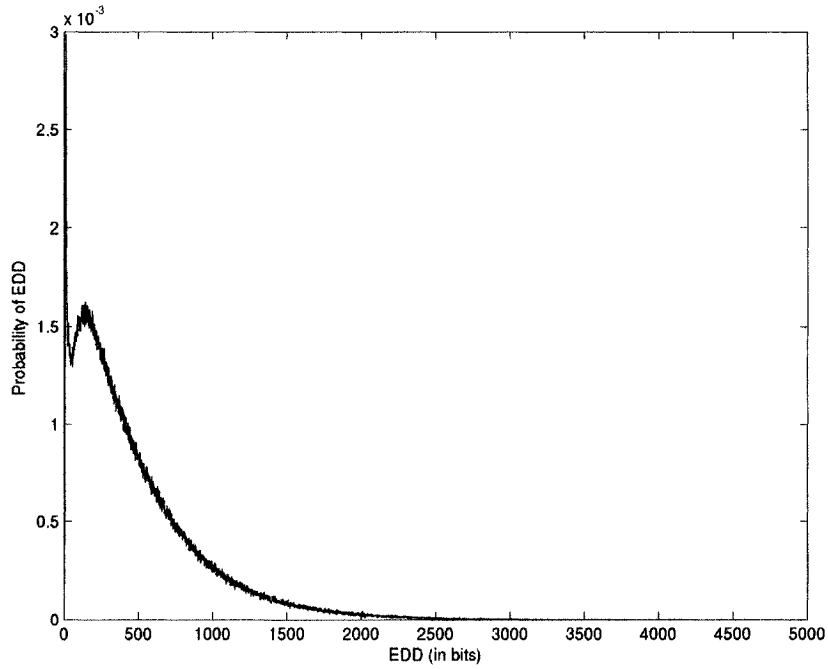

Figure 3.13: Histogram of EDD probability for the P-slices of video "Football"

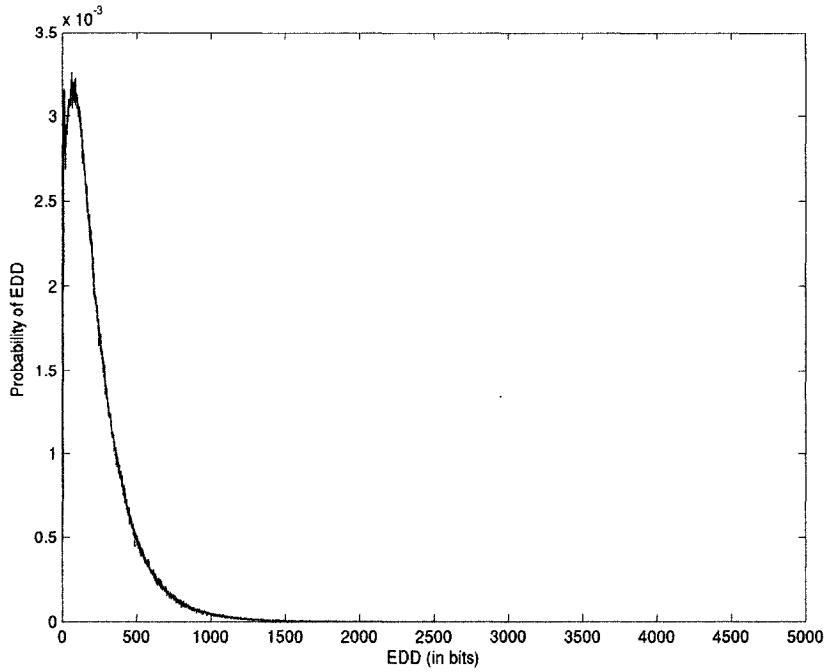

Figure 3.14: Histogram of EDD probability for the B-slices of video "Football"

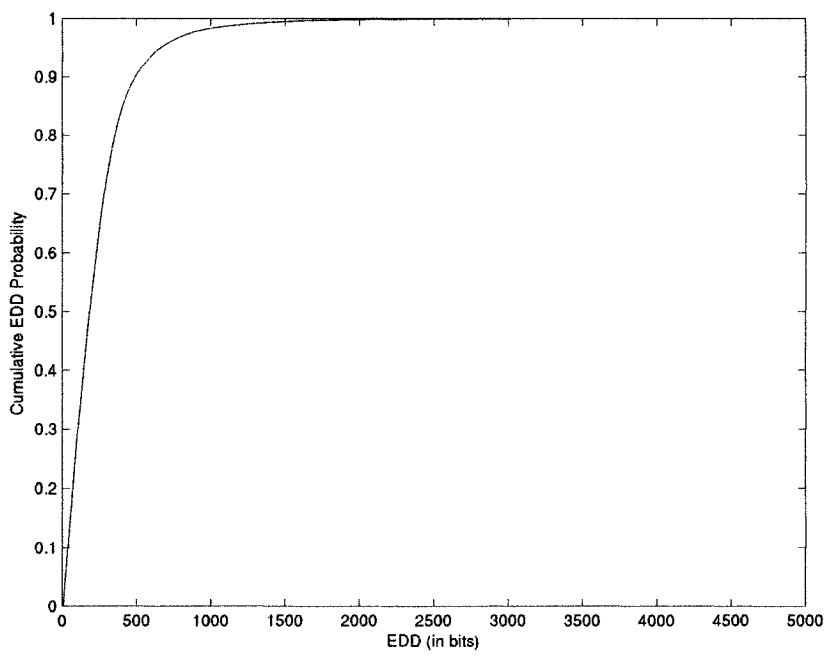

Figure 3.15: Histogram of cumulative EDD probability for the I-slices of video "Football"

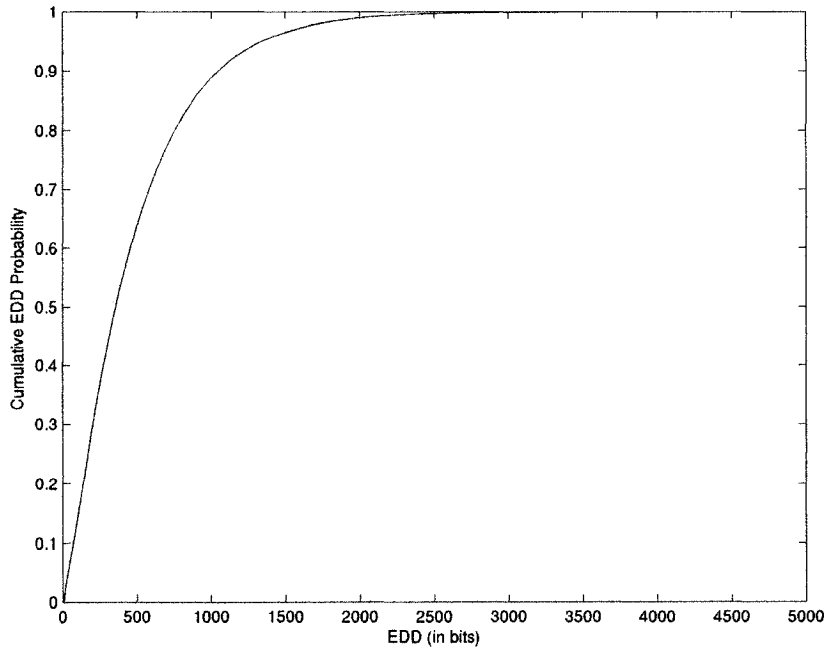

Figure 3.16: Histogram of cumulative EDD probability for the P-slices of video "Football"

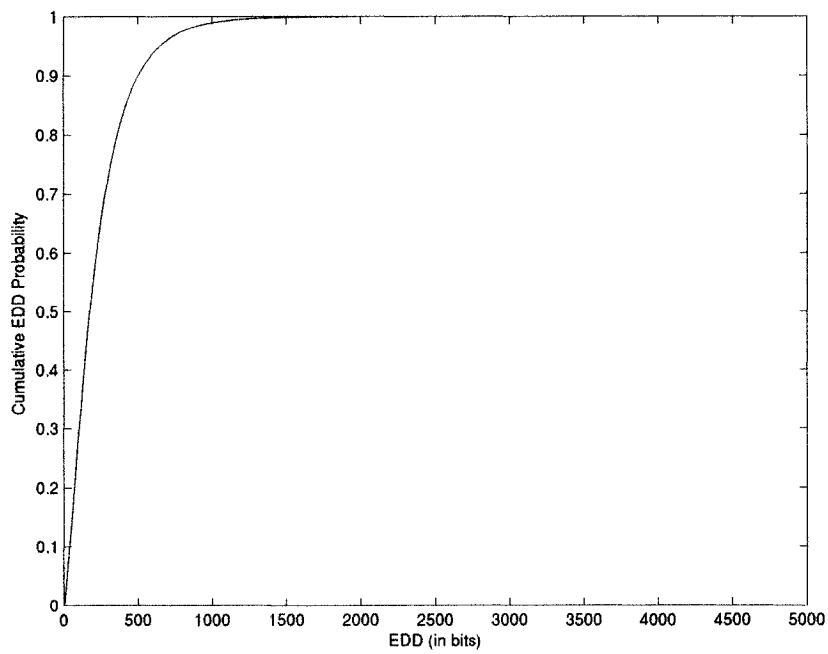

Figure 3.17: Histogram of cumulative EDD probability for the B-slices of video "Football"

3.4.2.2 Results for Video "Table-Tennis"

Figure 3.18 shows the EDD probability histogram for the video sequence "Table-Tennis" and Figure 3.19 shows the part of the histogram up to a delay of 400 bits. The curve for "Table-Tennis" has significant differences from the EDD histogram for "Football". There are three main observable differences. First, there is no second peak. After the initial spike, the curve drops and never increases. Second, frequency at the peak value is lower. Third, the tail end of the curve takes longer to approach zero.

Both histograms do, however, have peak values at 11 bits. Also, the EDD probability decreases and approaches zero as the delay increases for both curves.

Figure 3.20 shows the cumulative EDD probability. As compared to "Football", it can be observed that the knee point occurs significantly farther to the right. The knee point here occurs at around 1700 bits.

Figure 3.21, Figure 3.22, and Figure 3.23 show the EDD probability histograms for the I-slices, the P-slices and the B-slices respectively. It can be seen that, as in "Football", the histogram for the P-slices has a lower peak frequency and a longer tail than the I-slices and the B-slices.

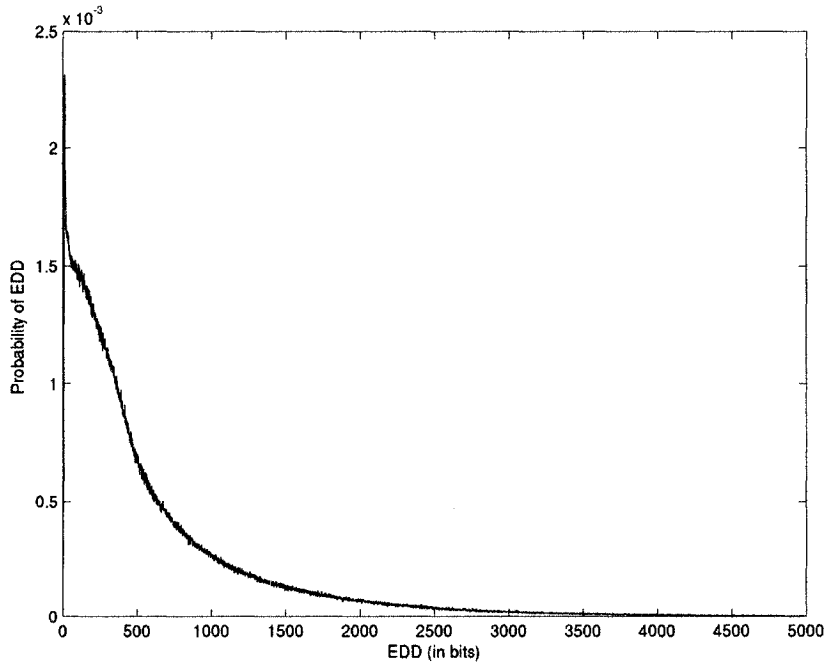

Figure 3.18: Histogram of EDD probability for video "Table-Tennis"

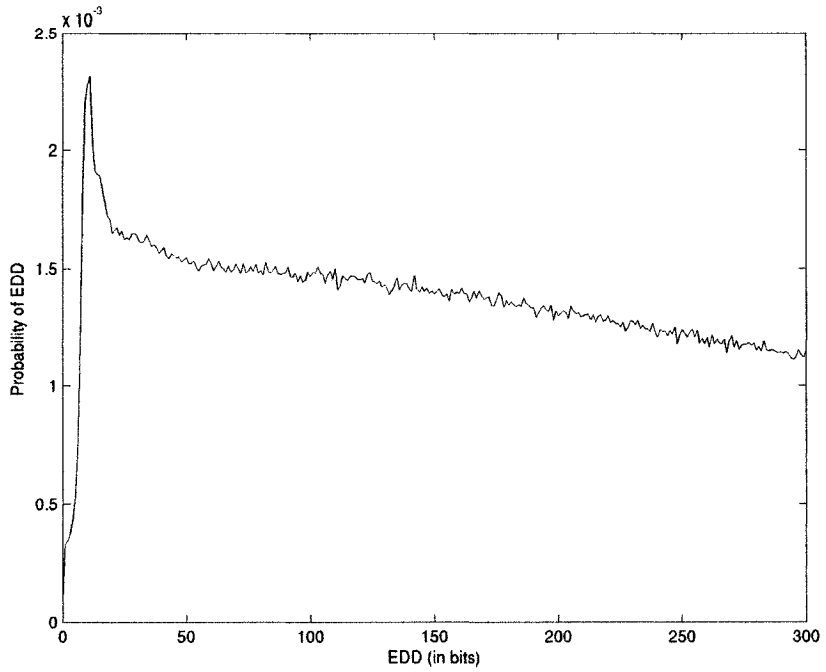

Figure 3.19: Segment of EDD histogram for video "Table-Tennis" up to a delay of 400 bits

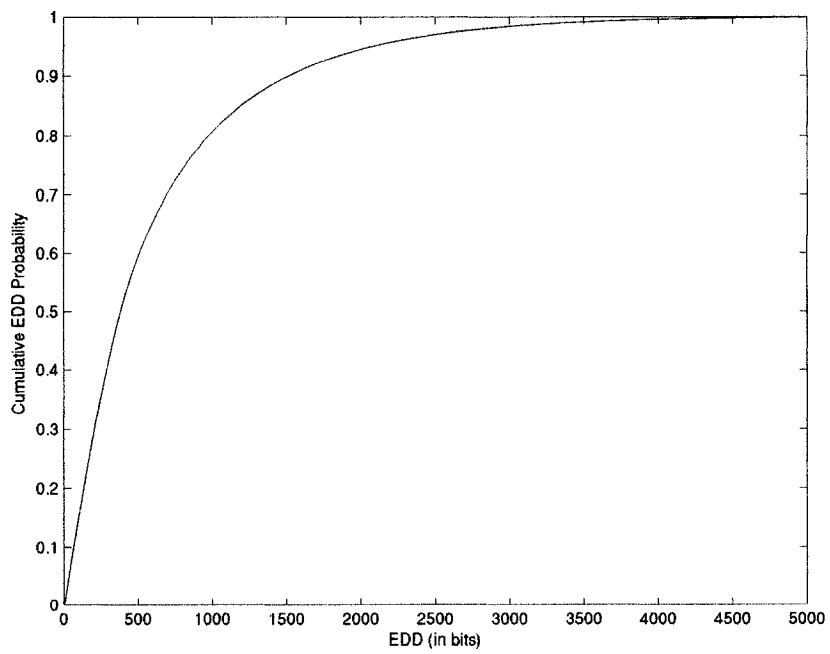

Figure 3.20: Cumulative EDD probability histogram for video "Table-Tennis"

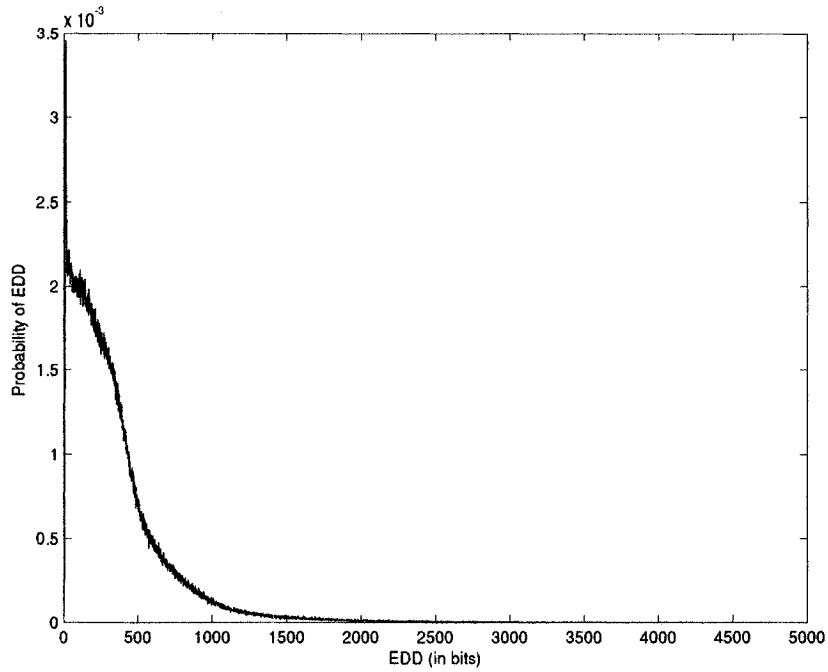

Figure 3.21: Histogram of EDD probability for the I-slices of video "Table-Tennis"

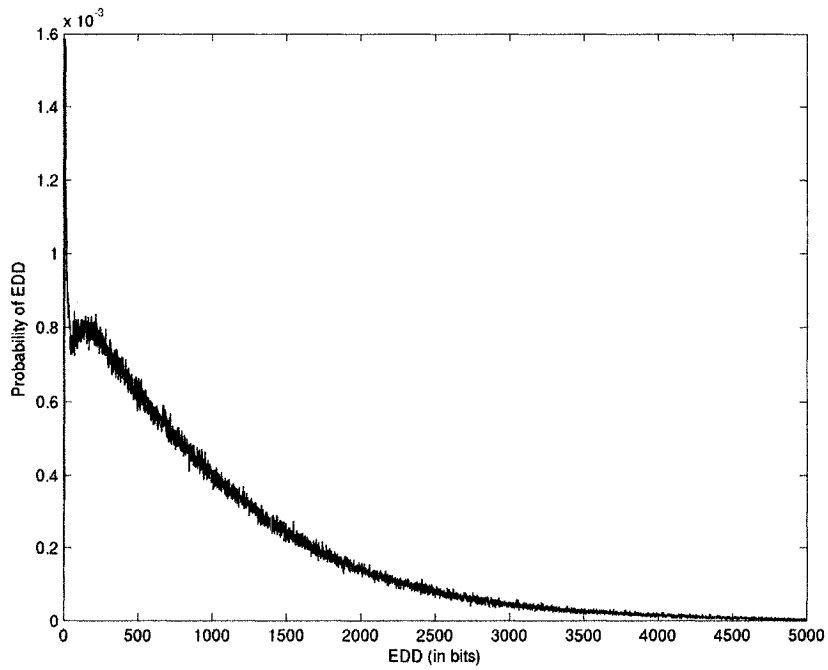

Figure 3.22: Histogram of EDD probability for the P-slices of video "Table-Tennis"

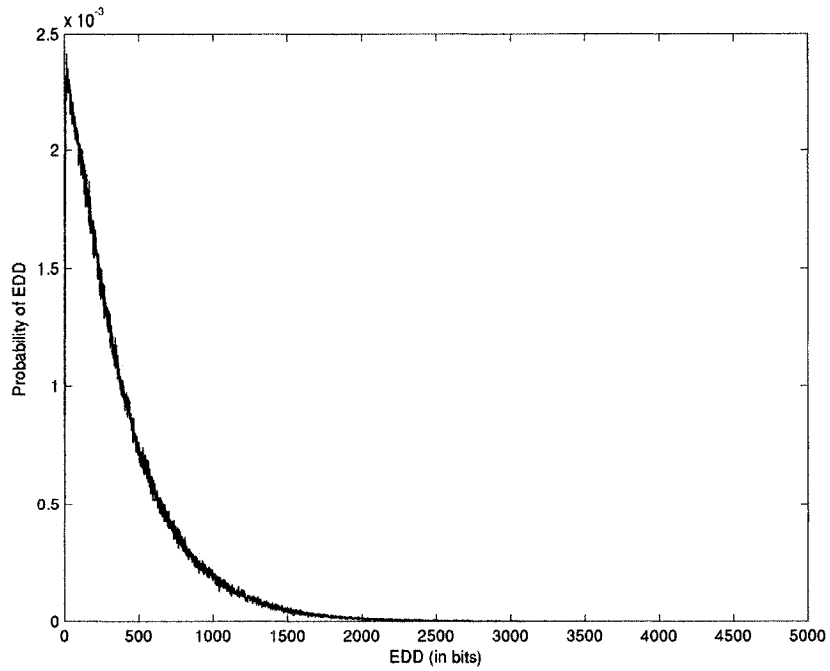

Figure 3.23: Histogram of EDD probability for the B-slices of video "Table-Tennis"

Figure 3.24, Figure 3.25, and Figure 3.26 show the cumulative EDD probability for the I-slices, the P-slices and the B-slices respectively. It can be observed that the knee point of the curve for the P-slices, as was the case for "Football", is farther to the right as compared to the knee points for the I-slices and the B-slices. However, the knee points for all three curves for "Table-Tennis" are further right in comparison to the corresponding curves for "Football". For instance, the knee point in the curve for the I-slices of "Football" is at around 550 bits whereas for "Table-Tennis" it is at around 800 bits.

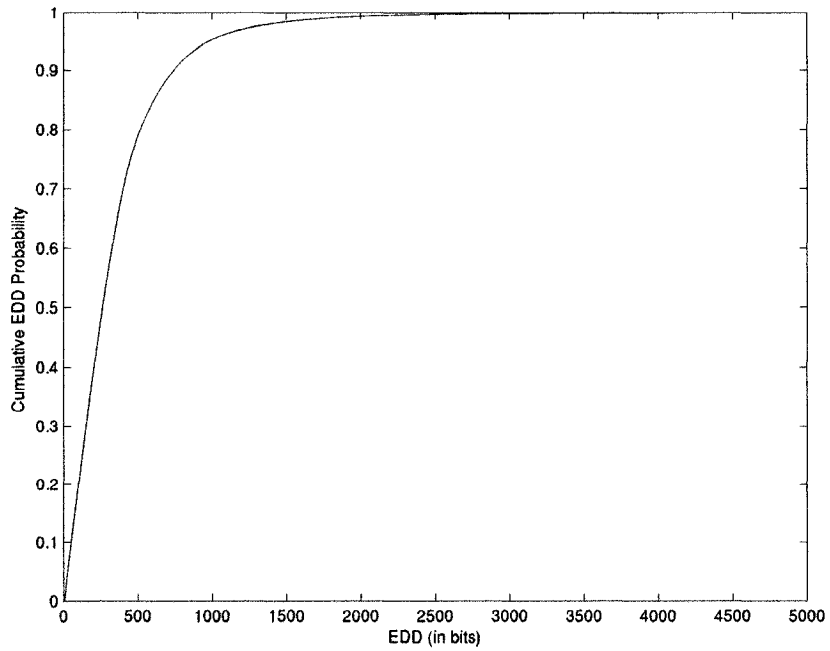

Figure 3.24: Cumulative EDD probability histogram for the I-slices of video "Table-Tennis"

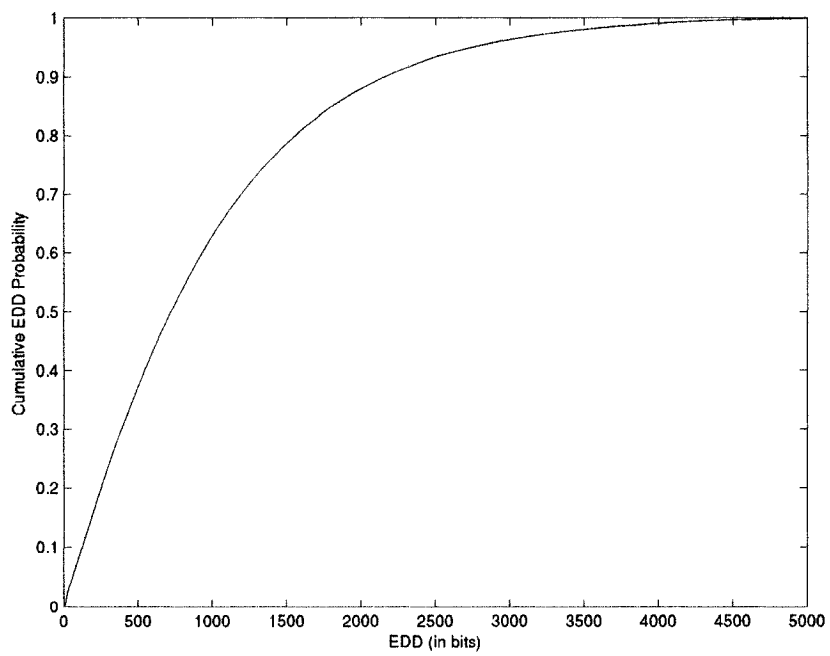

Figure 3.25: Cumulative EDD probability histogram for the P-slices of video "Table-Tennis"

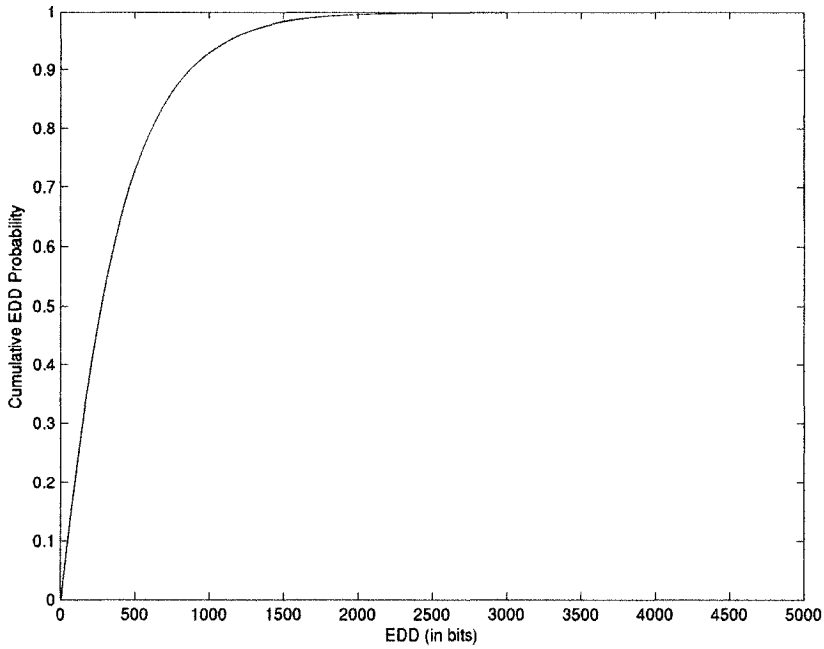

Figure 3.26: Cumulative EDD probability histogram for the B-slices of video "Table-Tennis"

3.4.3 Implications of Simulation Results

If there were an interval of bits in which the majority of the EDD values occurred, then this interval could be used to assist error correction by providing a guide as to where a bit error may be found. Based on the results, it appears that no such interval exists. This is due to the fact that the probability in the neighborhood of the peak is low and the tails of the curves are heavy.

The cumulative EDD probability could be used as a bit certainty parameter in an H.264 transmission scheme. Let d be a particular EDD value. The cumulative EDD probability at d is an estimate of the probability that the EDD is less than or equal to d. This is equivalent to the probability that a semantic error would be detected within d bits of a bit error. Let x be the detection bit of a particular slice. The cumulative EDD probability at d can thus be used to estimate the certainty of bit x - d.

There were significant differences between the cumulative probability estimates for the two video sequences tested. There were further differences observed between slice types. However, the general shape of all curves is similar. All of the cumulative

probability curves increase monotonically and have knee points. These general characteristics may be sufficient to estimate the bit certainty parameter.

3.5 Effect of Undetected Bit Errors

When a bit error does not cause a semantic error, it is known as an undetected bit error. Observations show that undetected bit errors do not cause any visible quality degradation in the output video. Furthermore, the PSNR of a frame containing an undetected bit error is never found to suffer a significant drop.

The effect of undetected bit errors on decompressed video was observed using the video sequences "Football" and "Table-Tennis". The luminance PSNR of several decompressed frames was observed. Each frame observed contained an undetected bit error. The frames were also examined to see if there was any noticeable degradation.

For the reader's own assessment, several frames with undetected bit errors are presented along with their corresponding PSNR values. The frames presented are Frames 1 and 58 of the video sequence and Frame 11 of the video sequence "Table-Tennis". Each frame contains an undetected bit error in one slice. Table 3.3 shows the luminance PSNR of the 3 frames and the maximum luminance PSNR of each frame, which is obtained when there are no bit errors in the frame and is determined by the compression. The PSNR of each of frame when it contains an undetected bit error is equal to the error-free PSNR.

Sequence & Frame #	PSNR (Y)	Maximum PSNR (Y)
Football, Frame 1	33.16 dB	33.16 dB
Football, Frame 58	28.10 dB	28.10 dB
Table-Tennis, Frame 11	32.88 dB	32.88 dB

Table 3.3: Luminance PSNR of frames with undetected bit errors

Figure 3.27 shows the error-free version of Frame 1 of "Football" and Figure 3.28 shows Frame 1 of "Football" with an undetected bit error. When comparing the two frames, there are no noticeable difference between them. Figure 3.29 to Figure 3.32 shows the same comparison for Frame 58 of "Football" and Frame 11 of "Table-Tennis".

Again, there are no visible differences between the error-free frames and the frames with undetected bit errors.

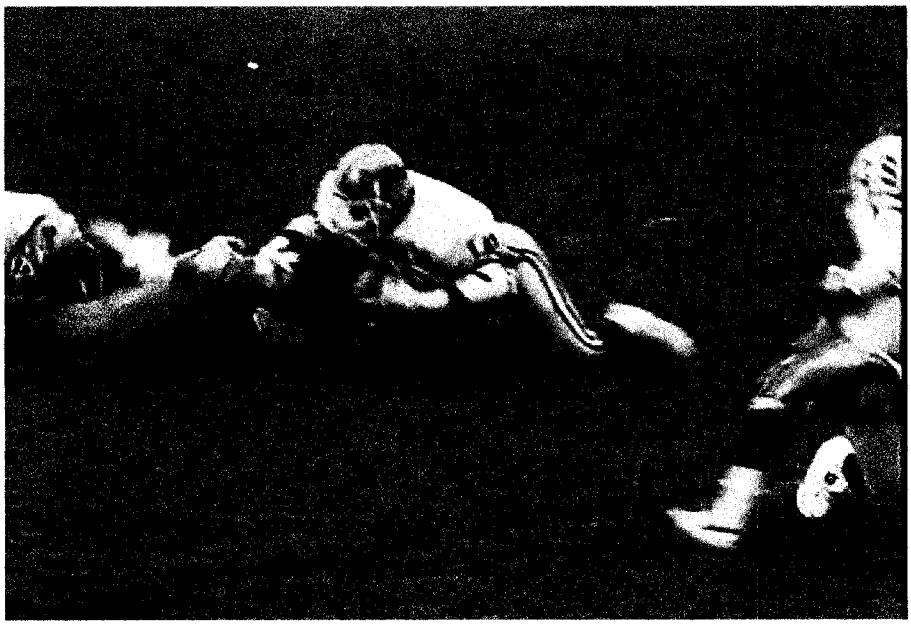

Figure 3.27: Frame 1 of error-free video sequence "Football"

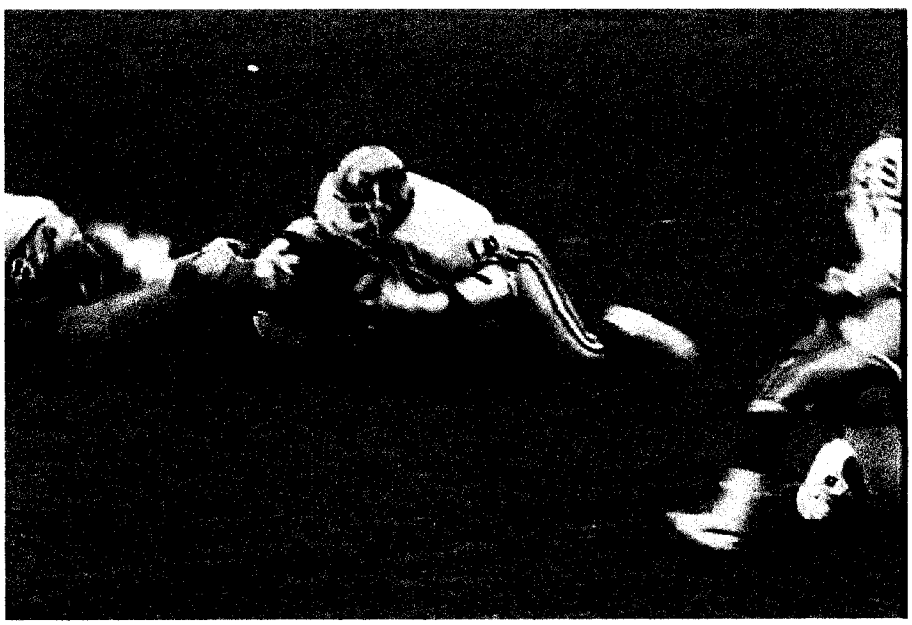

Figure 3.28: Frame 1 of video sequence "Football" with an undetected bit error

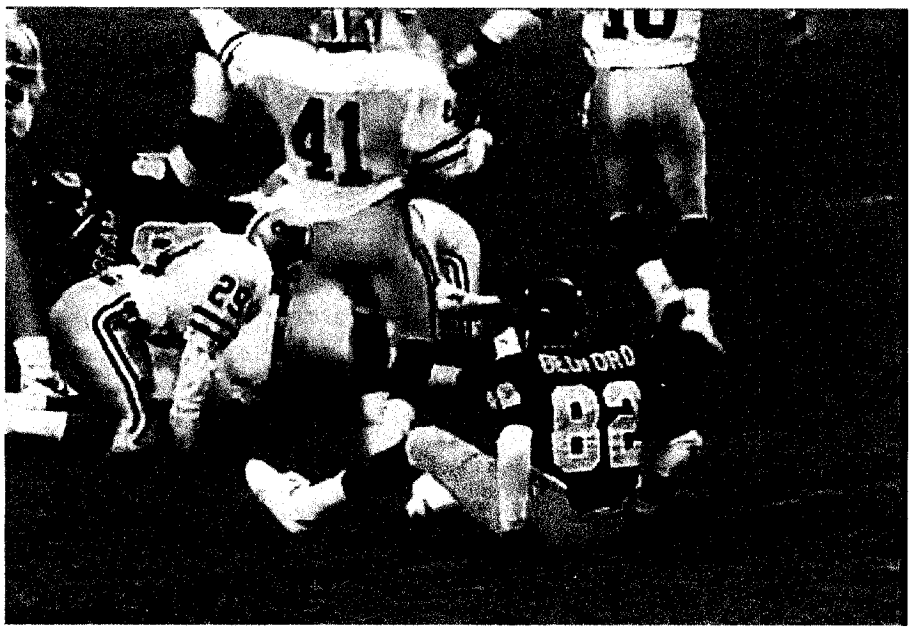

Figure 3.29: Frame 58 of error-free video sequence "Football"

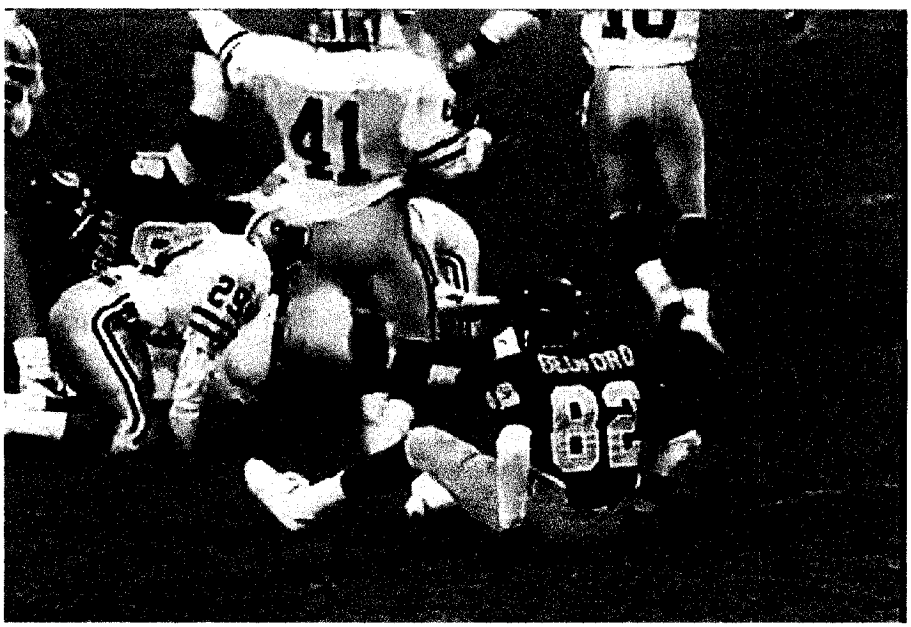

Figure 3.30: Frame 58 of video sequence "Football" with an undetected bit error

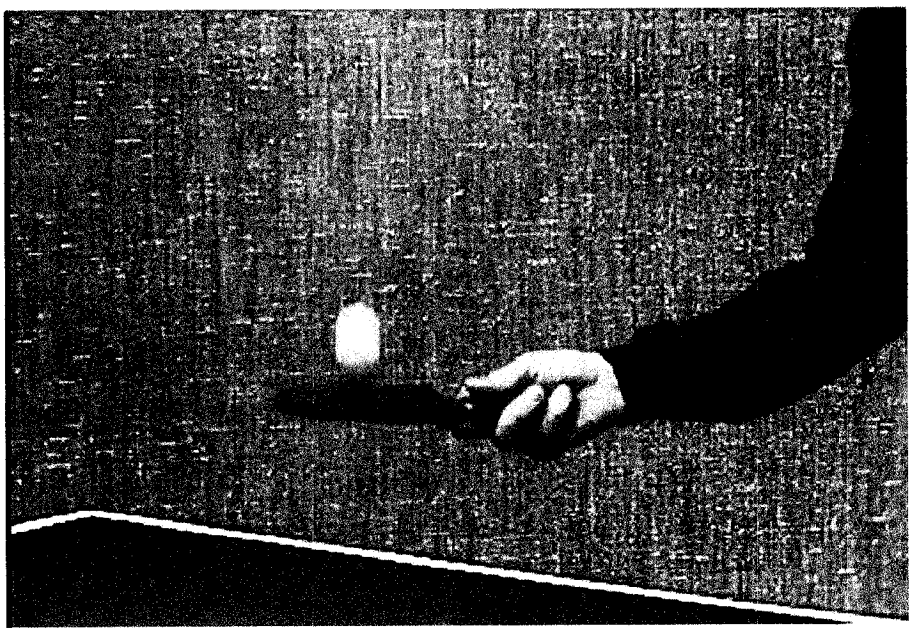

Figure 3.31: Frame 11 of error-free video sequence "Table-Tennis"

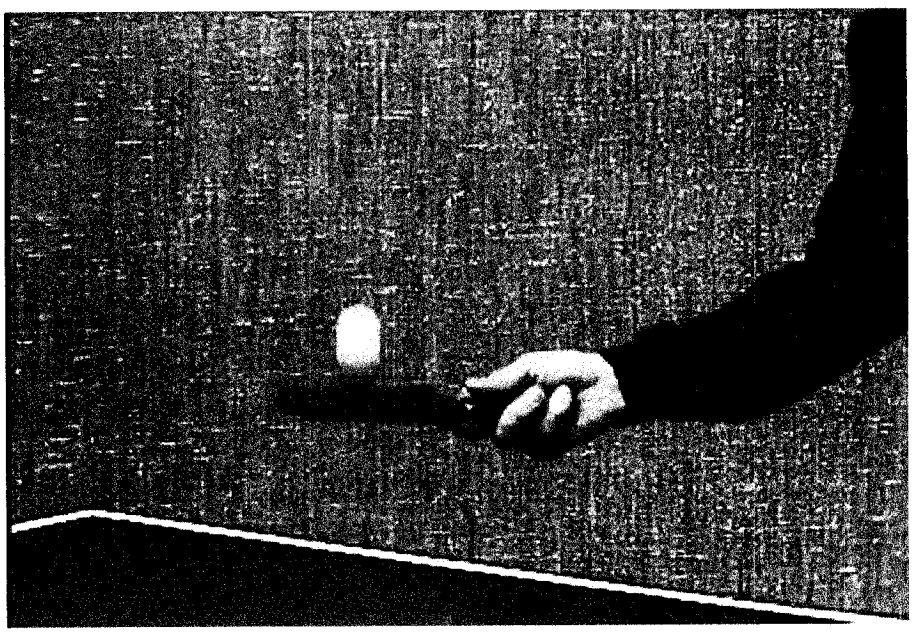

Figure 3.32: Frame 11 of video sequence "Table-Tennis" with an undetected bit error

3.6 Summary

This chapter discussed the effect of bit errors in H.264 as well as several types of detectable semantic errors. Three sets of simulations were performed in this chapter. The first set estimated the EDP in H.264. The results of these simulations showed that when a slice containing a bit error is decompressed, a semantic error is detected over 99% of the time. This implied that a slice with no semantic error rarely has any bit errors. The second set of simulations estimated the EDD probability. These simulations showed that short detection delays are more probable than long detection delays, but that long detection delays are not scarce. This implied that a bit error could not easily be located based solely on the location of a semantic error. The third set of simulations looked at the effect of undetected bit errors. These simulations showed that undetected bit errors do not severely degrade video quality.

This chapter has investigated features of error detection in H.264 that can be used in an H.264 transmission scheme to alleviate errors. The schemes proposed in Chapter 4 and Chapter 5 will use error detection to assist in correcting errors in an H.264 bitstream.

Chapter 4

Error Concealment using Source Semantics

Chapter 3 discussed the effects of bit errors on H.264 decompression and showed how semantic errors can be detected. It was found that bit errors lead to detectable semantic errors on average more than 99.3% of the time. It was also found that undetected bit errors do not appear to cause noticeable degradation. Because of this, the semantic correctness of a slice can reliably be used to assist in error concealment. This chapter proposes a decoding scheme that uses H.264 source semantics along with soft-channel information to conceal errors in a noisy video sequence.

In the current chapter, conventional error concealment methods are discussed in Section 4.1. A detailed description of the proposed scheme, Error Concealment using Source Semantics (ECSS), is presented in Section 4.2. The objective and subjective performance of the proposed scheme, as well as its complexity are evaluated in Section 4.3. Several design considerations for implementing the scheme in hardware are discussed in Section 4.4 and the chapter concludes with a summary in Section 4.5.

4.1 Conventional Error Concealment

When an error is detected in a compressed video slice, the decompressor usually moves on to the next slice even if it has not assigned values to all pixels in the current slice. As a result, some pixels may need to be filled in to display the video. The simplest way to fill missing pixels is to make them all the same color (usually black). While this is the easiest way to fill in missing values, it is not the most accurate. A better way to fill missing pixels is to estimate their values using available pixels in the neighborhood. This is known as conventional error concealment. Two common types of conventional error concealment methods are spatial error concealment and temporal error concealment [27].

Spatial error concealment reconstructs lost pixels by interpolating their values from neighboring pixels in the same frame. The simplest spatial error concealment method is linear interpolation. Several authors [28],[29] have proposed error concealment schemes that use linear interpolation. While linear interpolation conceals errors well in smooth areas, it often causes edges to be blurred. To address this, [30],[31] have proposed error concealment methods that use directional interpolation to improve edge concealment.

Temporal error concealment reconstructs lost pixels using pixels in neighboring frames and estimating the inter-frame motion [32],[33]. The simplest temporal error

concealment method is temporal replacement, which consists of filling in all missing pixels with the value of the pixel in the same spatial location in the previous frame. In other words, temporal replacement assumes that there is no motion between frames.

When the scheme proposed in this chapter fails to conceal an error, temporal replacement is used as a last resort. Temporal replacement by itself is also used as a benchmark for the proposed scheme.

4.2 Error Concealment using Source Semantics (ECSS)

The proposed scheme, Error Concealment using Source Semantics (ECSS), like several of the methods described in Section 2.3, uses source information to conceal errors in a noisy compressed sequence. The previous schemes operate on MPEG-2 and MPEG-4 video, which use variable length coding (VLC) entropy codes. In contrast, ECSS operates on H.264 video, which uses CABAC, an arithmetic entropy code. Also, ECSS does not require a channel code. As a result, it does not cause bandwidth expansion. ECSS generates slice candidates using soft-bit information in a similar manner to [39]. The block diagram of ECSS is seen in Figure 4.1.

On the transmitter side, the H.264 Compressor compresses the Input Video sequence. The Stream Splitter splits each compressed video slice into a Data Stream and a Header Stream. The Data Stream is BPSK modulated and transmitted over an AWGN channel, while the Header Stream is transmitted in a "side-channel" and is assumed to be error-free.

On the receiver side, the Stream Merger combines the two streams to produce a complete H.264 bitstream (with error-free header bits and noisy slice data bits). The Slice Candidate Generator (SCG) uses the soft values of the noisy slice data bits to generate slice candidates. The Source Semantic Verifier (SSV) verifies the semantics of the slice candidates and chooses a winner among them. The Winning Slice Candidate is decompressed by the H.264 Decompressor to produce the Output Video sequence.

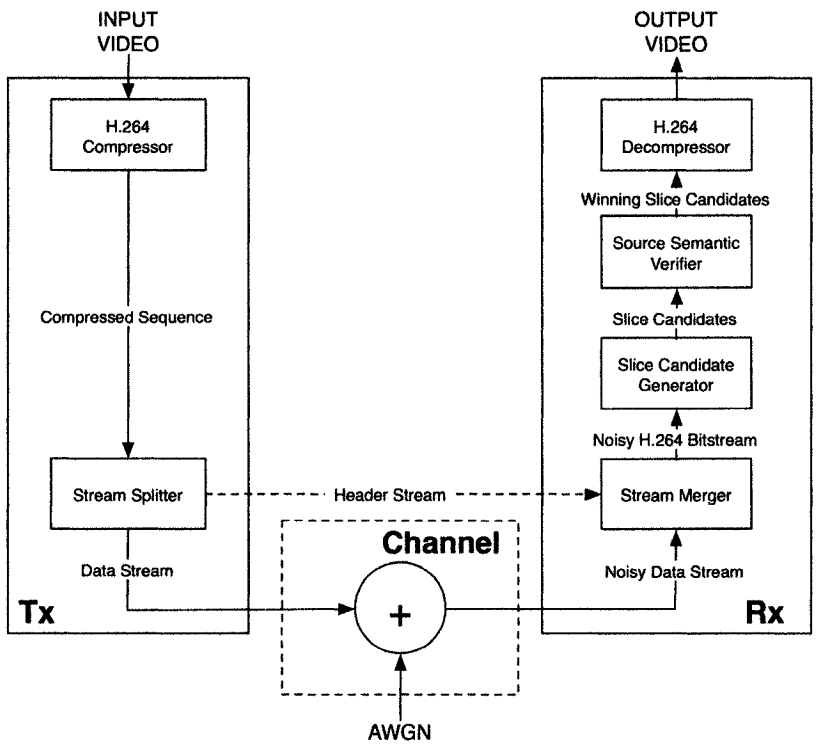

Figure 4.1: Block diagram for the proposed scheme Error Control using Source Semantics (ECSS)

The remainder of this section describes the modules in ECSS with the exception of the H.264 Compressor and H.264 Decompressor, which were discussed in Section 2.1.4 and Section 2.1.5 respectively. The Stream Splitter and the Stream Merger are described in Section 4.2.1. The Slice Candidate Generator is described in Section 4.2.2 and the Source Semantic Verifier is described in Section 4.2.3.

4.2.1 Stream Splitter and Stream Merger

In ECSS, as in the schemes discussed in Section 2.3, it is assumed that the only bits subject to errors are slice data bits. All bits in layers above the slice layer and all bits in slice headers (i.e. all other bits) are considered to be error-free.

The Stream Splitter separates an H.264 sequence into two bitstreams prior to transmission: the Data Stream and the Header Stream. The Data Stream contains all slice data bits and is transmitted over a normal AWGN channel. The Header Stream contains all other bits and is protected by a very strong coding scheme so that error-free transmission can be assumed. The length of each slice is appended to the end of each slice header in the Header Stream so that the ECSS receiver can merge the two streams.

On the receiver side, the Stream Merger recombines the noisy Data Stream with the error-free Header Stream to produce a complete but noisy H.264 bitstream.

4.2.2 Slice Candidate Generator

For each video slice, the Slice Candidate Generator (SCG) generates a list of slice candidates to undergo semantic verification (by the Source Semantic Verifier, described in Section 4.2.3). A slice candidate is defined as a bit sequence that could possibly be the original slice. If a slice is N bits long, then there are 2^N possible slice candidates (i.e. all bit sequences of length N). Among the 2^N slice candidates, one candidate, called the Target Candidate, is identical to the original slice. The goal of the SCG is to include the Target Candidate as high as possible on the list of slice candidates.

Given the received soft values, some slice candidates are more likely to be the Target Candidate than others. Using the soft values, a measure can be derived to rank slice candidates in descending order of likelihood. When the amount of channel noise is low, this measure will usually rank the Target Candidate high on the list. However, as the amount of channel noise increases, the Target Candidate will often occur lower down.

To ensure that the Target Candidate is in the list, the SCG could theoretically generate all 2^N possible slice candidates. However, for practical values of N, it is too complex to verify the semantics of 2^N slice candidates. Therefore, the SCG only generates the n_{sc} most likely slice candidates where n_{sc} is a number chosen to yield reasonable complexity. Because the list only contains n_{sc} slice candidates, it might not contain the Target Candidate, although this usually occurs only when the amount of channel noise is high.

There are two tasks required to develop the SCG. First, the measure to rank the slice candidates in descending order of likelihood is determined in Section 4.2.2.1. Then, an algorithm uses the measure to determine the n_{sc} most likely slice candidates. This algorithm is called the Incomplete Partial Sums Algorithm (IPSA) and is discussed in Section 4.2.2.2.

4.2.2.1 *Determination of Measure to Rank Slice Candidates*

For a given slice of length N, the transmitted bit sequence, denoted **u**, is:

$$\mathbf{u} = [u_0, u_1, ..., u_{N-1}] \ , \ \mathbf{u} \in [0,1]^N \tag{4.1}$$

Here, u_k is a transmitted bit, and $k = 0,1,...N-1$. After transmission, the received sequence of soft values, denoted **y**, is:

$$\mathbf{y} = [y_0, y_1, ..., y_{N-1}] \ , \ \mathbf{y} \in \Re^N \tag{4.2}$$

Here, y_k is a received soft value, and $k = 0,1,...N-1$. Using the received sequence **y**, slice candidates are generated. A particular slice candidate, denoted **s**, is:

$$\mathbf{s} = [s_0, s_1, ..., s_{N-1}] \ , \ \mathbf{s} \in [0,1]^N \tag{4.3}$$

Here, s_k is a bit in the slice candidate, and $k = 0,1,...N-1$. The likelihood of bit s_k is the probability that s_k is correct and is denoted $L(s_k)$:

$$L(s_k) = P(s_k = u_k \mid y_k) \tag{4.4}$$

Refer to Section 2.2 for details. The likelihood of slice candidate **s**, denoted $L(\mathbf{s})$ is defined as the product of the likelihood of each of its bits:

$$L(\mathbf{s}) = \prod_{k=0}^{N-1} L(s_k) = \prod_{k=0}^{N-1} P(s_k = u_k \mid y_k) \tag{4.5}$$

Slice candidates are ranked in descending order of likelihood $L(\mathbf{s})$. However, calculating each value of $L(\mathbf{s})$ requires evaluating the likelihood of all bits in the slice candidate, which could be time consuming. The calculation can be reduced because the SCG can identify the slice candidates that have the highest likelihood without actually calculating $L(\mathbf{s})$. A new measure is thus derived.

For each slice, there is one slice candidate with the highest likelihood. This slice candidate, labeled **v**, is called the Primary Slice Candidate (PSC). The bits in the PSC are determined by performing a hard decision on each of the received soft values:

$$\mathbf{v} = [v_0, v_1, ..., v_{N-1}] \ , \ v_k = \begin{cases} 0, & y_k < 0 \\ 1, & y_k \geq 0 \end{cases} \tag{4.6}$$

Here, v_k is a bit in the PSC, and $k = 0,1,...N-1$. The likelihood of each slice candidate is divided by the likelihood of the PSC:

$$\frac{L(\mathbf{s})}{L(\mathbf{v})} = \prod_{k=0}^{N-1} \frac{P(s_k = u_k \mid y_k)}{P(v_k = u_k \mid y_k)} \tag{4.7}$$

The logarithm of Equation 4.7 is then taken:

$$\log\left(\frac{L(\mathbf{s})}{L(\mathbf{v})}\right) = \log\left(\prod_{k=0}^{N-1} \frac{P(s_k = u_k \mid y_k)}{P(v_k = u_k \mid y_k)}\right) = \sum_{k=0}^{N-1} \log\left(\frac{P(s_k = u_k \mid y_k)}{P(v_k = u_k \mid y_k)}\right) \tag{4.8}$$

Finally, the value in Equation 4.8 is multiplied by $-\sigma^2/2$, where σ^2 is the variance of the channel noise. This produces a new value, as shown in Equation 4.9. This value, denoted $R(\mathbf{s})$, is called the rank of a slice candidate.

$$R(\mathbf{s}) = -\frac{\sigma^2}{2}\log\left(\frac{L(\mathbf{s})}{L(\mathbf{v})}\right) = -\frac{\sigma^2}{2}\sum_{k=0}^{N-1} \log\left(\frac{P(s_k = u_k \mid y_k)}{P(v_k = u_k \mid y_k)}\right) \tag{4.9}$$

As the negation reverses the relative order of slice candidates, ranking the slice candidates in ascending order of $R(\mathbf{s})$ puts them in descending order of likelihood. Thus a lower $R(\mathbf{s})$ means the slice candidate is more likely and a higher $R(\mathbf{s})$ means the candidate is less likely.

Usually, the PSC **v** and slice candidate **s** only differ by a few bits. The bits that are different between **v** and **s** are called flip-bits. The bits they have in common are non-flip-bits. The sum in Equation 4.9 can be separated into a part for flip-bits and a part for non-flip-bits:

$$R_s = -\frac{\sigma^2}{2}\left[\sum_{k \notin F_s} \log\left(\frac{P(s_k = u_k \mid y_k)}{P(v_k = u_k \mid y_k)}\right) + \sum_{k \in F_s} \log\left(\frac{P(s_k = u_k \mid y_k)}{P(v_k = u_k \mid y_k)}\right)\right] \tag{4.10}$$

Here, F_s is the set of indices to the flip bits of **s** and is represented as:

$$k \in F_s \text{ if } s_k \oplus v_k = 1 \tag{4.11}$$

In Equation 4.11, $\oplus$ is the exclusive-or. If there are M flip-bits in **s**, F_s is written as:

$$F_s = \{k_1, k_2, ..., k_M\} \tag{4.12}$$

Using Equation 4.11, each of the terms in Equation 4.10 can be simplified separately. For any given bit k:

$$s_k = \begin{cases} v_k, & k \notin F_s \\ v_k', & k \in F_s \end{cases} \tag{4.13}$$

Here, v_k' is the opposite of v_k (i.e. v_k' is 1 when v_k is 0 and vice versa). Equation 4.10 becomes:

$$R(\mathbf{s}) = -\frac{\sigma^2}{2}\left[\sum_{k\notin F_s} \log\left(\frac{P(v_k = u_k \mid y_k)}{P(v_k = u_k \mid y_k)}\right) + \sum_{k\in F_s} \log\left(\frac{P(v_k' = u_k \mid y_k)}{P(v_k = u_k \mid y_k)}\right)\right] \tag{4.14}$$

The first summation is equal to 0, thus Equation 4.14 reduces to:

$$R(\mathbf{s}) = -\frac{\sigma^2}{2}\sum_{k\in F_s} \log\left(\frac{P(v_k' = u_k \mid y_k)}{P(v_k = u_k \mid y_k)}\right) \tag{4.15}$$

The likelihood values in Equation 4.15 are now evaluated using Equation 2.4. For BPSK transmission over an AWGN channel, the likelihood of bit v_k is determined as:

$$P(v_k = u_k \mid y_k) = \begin{cases} \dfrac{1}{1+e^{2y_k/\sigma^2}}, & y_k < 0 \\ \dfrac{e^{2y_k/\sigma^2}}{1+e^{2y_k/\sigma^2}}, & y_k \geq 0 \end{cases} \tag{4.16}$$

$$P(v_k' = u_k \mid y_k) = \begin{cases} \dfrac{e^{2y_k/\sigma^2}}{1+e^{2y_k/\sigma^2}}, & y_k < 0 \\ \dfrac{1}{1+e^{2y_k/\sigma^2}}, & y_k \geq 0 \end{cases} \tag{4.17}$$

Using Equations 4.16 and 4.17, each element of the sum in Equation 4.15 can be written as:

$$\begin{aligned} \log\left(\frac{P(v_k' = u_k \mid y_k)}{P(v_k = u_k \mid y_k)}\right) &= \begin{cases} \log(e^{2y_k/\sigma^2}), & y_k < 0 \\ \log(e^{-2y_k/\sigma^2}), & y_k \geq 0 \end{cases} \\ &= \begin{cases} 2y_k/\sigma^2, & y_k < 0 \\ -2y_k/\sigma^2, & y_k \geq 0 \end{cases} \\ &= -\left|2y_k/\sigma^2\right| \end{aligned} \tag{4.18}$$

Thus, Equation 4.15 reduces to:

$$\begin{aligned} R(\mathbf{s}) &= -\frac{\sigma^2}{2}\sum_{k\in F_s} -\left|2y_k/\sigma^2\right| \\ &= \sum_{k\in F_s}\left|y_k\right| \end{aligned} \tag{4.19}$$

In short, the rank $R(\mathbf{s})$ of slice candidate $\mathbf{s}$ is defined only by the sum of the magnitudes of the soft values of its flip bits. Intuitively this makes sense because the flip bits in the most likely slice candidates should be the least certain bits, and the bits with the lowest magnitude of y_k are the ones that are least certain. This supports the simplicity of Equation 4.19. Since the PSC $\mathbf{v}$ has no flip-bits, $R(\mathbf{v})$ will always be zero and hence it will always be the most likely slice candidate.

4.2.2.2 The Incomplete Partial Sums Algorithm (IPSA)

The SCG generates the n_{sc} most likely slice candidates and puts them in ascending order of $R(\mathbf{s})$ (Equation 4.19), which puts them in descending order of likelihood. The list of the n_{sc} smallest $R(\mathbf{s})$ values, denoted $\mathbf{b}$, is called the Output List and is written as:

$$\mathbf{b} = [b_0, b_1, \ldots, b_{n_{sc}-1}], \quad \mathbf{b} \in \Re^{n_{sc}} \tag{4.20}$$

Here, b_k is an Output Element, and $k = 0,1,\ldots,n_{sc}-1$. Each Output Element is the $R(\mathbf{s})$ value corresponding to a slice candidate. As $R(\mathbf{s})$ values are calculated, the SCG keeps track of the corresponding slice candidates. When the Output List is complete, the SCG records the actual slice candidates and not the $R(\mathbf{s})$ values.

For a slice of length N, one way to generate the Output List $\mathbf{b}$ is to produce all 2^N $R(\mathbf{s})$ values, sort them and choose the first n_{sc} values as Output Elements. However, the complexity of such a task is of order $2^N \cdot \log_2(2^N)$ when using a fast sorting algorithm, such as Quicksort [34]. For typical values of N, this complexity is too high to be practical. Because only the first n_{sc} values are needed, the SCG uses a more efficient algorithm called the Incomplete Partial Sums Algorithm (IPSA), which has a more reasonable complexity.

The IPSA first sorts the received slice $\mathbf{y}$ in order of the magnitude of its elements. The sorted elements are used to generate the $R(\mathbf{s})$ values. The IPSA then iteratively adds $R(\mathbf{s})$

values to the Output List, ensuring that each $R(\mathbf{s})$ it adds is the smallest value not currently in the Output List. The IPSA stops after n_{sc} iterations.

The IPSA begins by sorting the absolute values of the soft-valued bits in received slice **y**. This produces **a**, the Input List, which is written as:

$$\mathbf{a} = [a_0, a_1, ..., a_{M-1}], \quad \mathbf{a} \in \Re^M \tag{4.21}$$

Here, a_j is called an Input Element where $j = 0,1,...M-1$. The Input Elements are $|y_k|$ values (where $k = 0,1,...N-1$) in ascending order. M is the size of the Input List:

$$M = \begin{cases} n_{sc}, & N > n_{sc} \\ N, & N < n_{sc} \end{cases} \tag{4.22}$$

The value of M in Equation 4.22 is determined as the minimum number of Input Elements needed to generate the Output List. When each Output Element has exactly one flip-bit, n_{sc} Input Elements are needed. If any Output Element has two or more flip-bits, its flip-bits will all occur in at least one other Output Element. Clearly, less than n_{sc} Input Elements are needed in this case. To handle all cases, the IPSA puts n_{sc} elements in the Input List unless $N < n_{sc}$, in which case it puts all N elements in the Input List.

To create the Input List, the IPSA uses the Heapsort algorithm to sort the magnitudes of the values in **y** [34]. The algorithm stops as soon as M values are sorted. The complexity of this task is $M \cdot \log(N)$.

Each Input Element corresponds to the absolute value of a received soft value. The value of $R(\mathbf{s})$ in Equation 4.19 can therefore be written as a sum of Input Elements, as in Equation 4.23. In Equation 4.23, G_s is the set of indices to the Input Elements that correspond to the flip-bits of slice candidate **s** and a_k is a particular Input Element.

$$R(\mathbf{s}) = \sum_{k \in G_s} a_k \tag{4.23}$$

Once the Input List is created, the IPSA builds a min-heap. The heap is used to store the $R(\mathbf{s})$ values that could be the next Output Element. To begin the IPSA, the Output List and the heap each contain one element. The Output List contains $R(\mathbf{v})$, and the heap contains the slice candidate with $G_s = \{0\}$ (because it always has the next smallest $R(\mathbf{s})$).

On any iteration, the root value in the heap is $R(\mathbf{x})$, the rank of slice candidate **x**, and is moved to the Output List. Two new slice candidates, denoted **y** and **z**, are then generated. Their corresponding ranks, $R(\mathbf{y})$ and $R(\mathbf{z})$, are then added to the heap.

The first new slice candidate, **y**, has the same number of flip-bits as **x**. Equation 4.24 shows how to get $R(\mathbf{y})$ from $R(\mathbf{x})$. Here, a_r is the largest Input Element in the sum of $R(\mathbf{x})$ and a_{r+1} is the next largest Input Element after a_r.

$$R(\mathbf{y}) = R(\mathbf{x}) - a_r + a_{r+1} \tag{4.24}$$

The second new slice candidate, **z**, has one more flip-bit than **x**. Equation 4.25 shows how to get $R(\mathbf{z})$ from $R(\mathbf{x})$.

$$R(\mathbf{z}) = R(\mathbf{x}) + a_{r+1} \tag{4.25}$$

After $R(\mathbf{y})$ and $R(\mathbf{z})$ are placed in the heap, they are sifted into it. After sifting, the value on top of the heap is the smallest value in the heap and is R(**x**) for the next iteration.

If a_r is the largest Input Element in the Input List, then a_{r+1} does not exist and neither **y** nor **z** can be produced. In this case, no new values are added to the heap, and the heap size decreases by 1. This can only occur when $n_{sc} > N$.

The IPSA keeps the size of the heap small while ensuring that it always contains the next Output Element. The example that follows illustrates how the IPSA guarantees that the heap contains the next Output Element.

Example: The SCG receives a soft valued sequence **y**. When $|y_k|$, the magnitudes of the soft values are sorted in ascending order, the resulting Input List **a** is:

$$\mathbf{a} = [1.1,\ 2.8,\ 4.2,\ 6.7,\ 7.2,\ 7.3,\ 7.9,\ \ldots] \tag{4.26}$$

Figure 4.2 shows the first few iterations of the IPSA. Each value in the Output List and in the heap are calculated using Equation 4.23. Each iteration, $R(\mathbf{x})$ is moved from the top of the heap to the Output List and $R(\mathbf{y})$ and $R(\mathbf{z})$ (calculated using Equations 4.24 and 4.25) and added to the heap. Clearly the next Output Element is always one of the values in the heap.

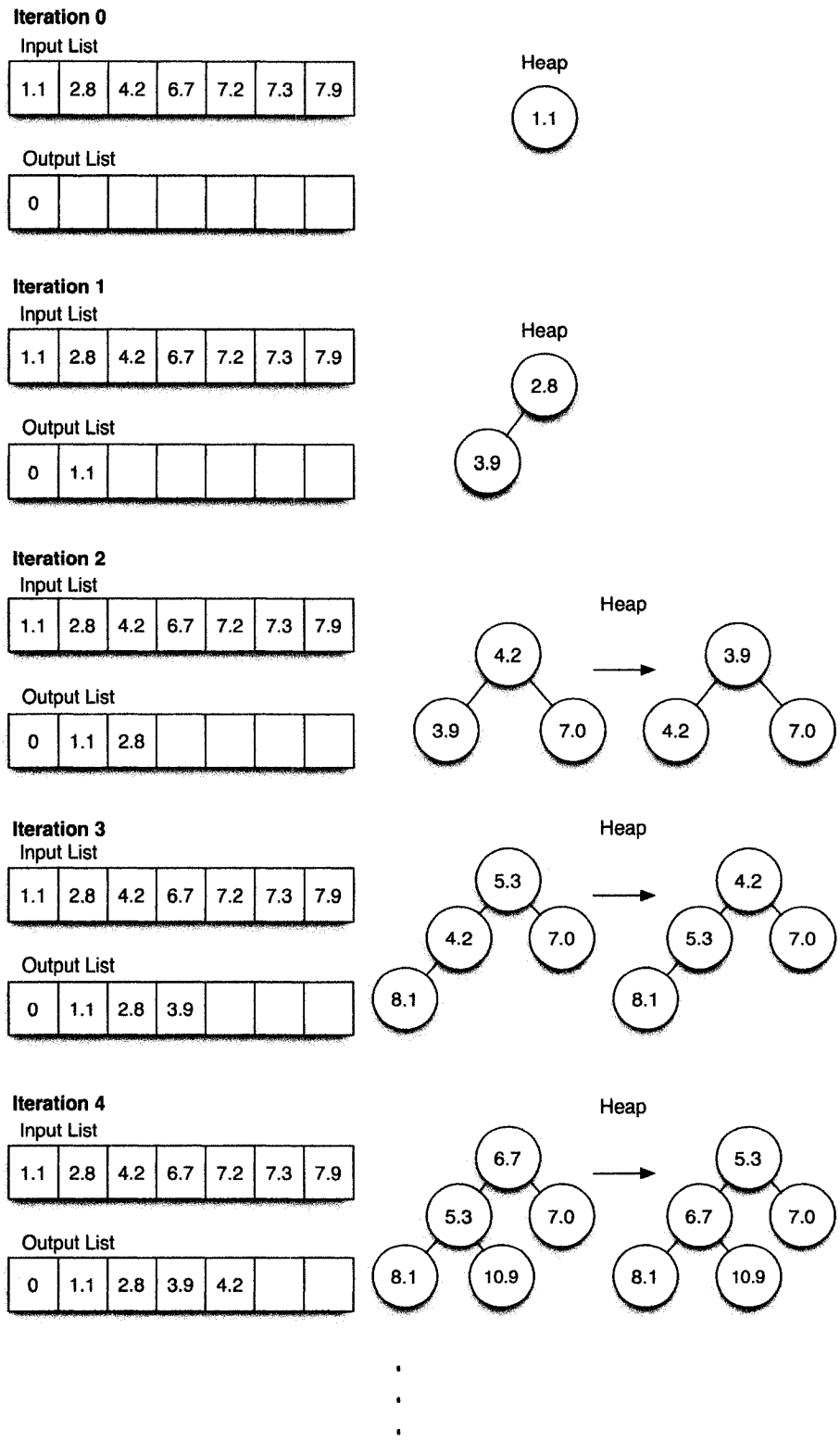

Figure 4.2: First few iterations of the Incomplete Partial Sums Algorithm

The IPSA terminates either when the heap size reaches zero or when the output list reaches n_{sc} elements. The former occurs when $n_{sc} > 2^N$. The latter occurs when $n_{sc} < 2^N$.

The complexity of the IPSA can be calculated analytically. There are two parts to the IPSA: the creation of the Input List and the creation of the Output List.

The complexity of the Heapsort that creates the Input List is $n_{sc} \log N$ where n_{sc} is the number of slice candidates generated and N is the length of the slice.

When generating the Output List, the main complexity comes from sifting the two slice candidates into the heap on each iteration. If there are n elements in the heap, the complexity of the sifting operation is $\log n$. Because the algorithm proceeds for n_{sc} iterations and the maximum heap size is n_{sc}, the complexity of all iterations is less than $n_{sc} \log n_{sc}$.

When $N > n_{sc}$, The complexity of the IPSA is dominated by the creation of the Input List. Thus the overall complexity is $n_{sc} \log N$. On the other hand, when $N < n_{sc}$, the complexity is dominated by the generation of the Output List and is less than $n_{sc} \log n_{sc}$.

4.2.3 Source Semantic Verifier

The Source Semantic Verifier (SSV) checks the primary slice candidate of each slice for semantic errors. If a semantic error is detected, the SSV goes down the list of slice candidates generated by the SCG and tests them, one by one, until it finds a slice candidate that does not cause a semantic error or until it has tested all slice candidates generated.

The decoding path of any slice candidate only begins to diverge from the decoding path of the PSC at the first flip-bit. To take advantage of this property, the SSV records its parameters at the start of each macroblock as it verifies the PSC. When verifying subsequent slice candidates, the SSV determines which macroblock contains the first flip-bit and restarts testing from the start of that macroblock (by recalling the appropriate parameters). Semantic verification proceeds from that point rather than from the beginning of the slice. Clearly, this reduces the time required to verify a slice candidate.

A slice candidate passes semantic verification when no semantic error is detected. The first candidate to pass semantic verification is declared the winning candidate and the SSV stops checking further slice candidates. In contrast, a slice candidate fails semantic

verification when a semantic error is detected. When this occurs, the SSV records the position of the bit at which the semantic error is detected. This position is referred to as the Detection Bit Position (DBP). The SSV then moves on to the next slice candidate. In the event that all slice candidates fail semantic verification, the candidate with the largest DBP (i.e. the slice where the first semantic error was detected latest) is chosen as the winning candidate.

The winning candidate is passed to the H.264 Decompressor where it is finally decompressed.

4.3 Performance Evaluation

The performance of the proposed error concealment scheme is compared to the performance of temporal concealment. Performance is measured both objectively, using several objective measures, and subjectively, by looking at several decompressed video sequences. For the reader's own assessment, the subjective performance evaluation includes a few sample frames that are decompressed by both the proposed scheme and the temporal concealment scheme. The complexity of the proposed scheme is also measured and discussed.

4.3.1 Selection of Winning Slice Candidate

As discussed in Section 4.2.3, when all slice candidates fail semantic verification, the SSV selects the candidate that has the largest detection bit as the winning candidate. This slice candidate is chosen because it is believed that less video data gets lost as more bits in a slice are read. In contrast, the scheme in [39] selects the PSC as the winning slice candidate when no valid slice candidates were found.

To verify which method yields better results, a set of simulations was performed. The proposed scheme was modified to select the PSC as the winning candidate when all candidates fail semantic verification. The results of this modified scheme were measured at various channel SNR values and were compared to the results of the proposed scheme. The simulations were run on the video sequence "Table-Tennis" and a simulation run was carried out 10 times with independent seeds at each channel SNR. For all simulations, the schemes used 300 slice candidates.

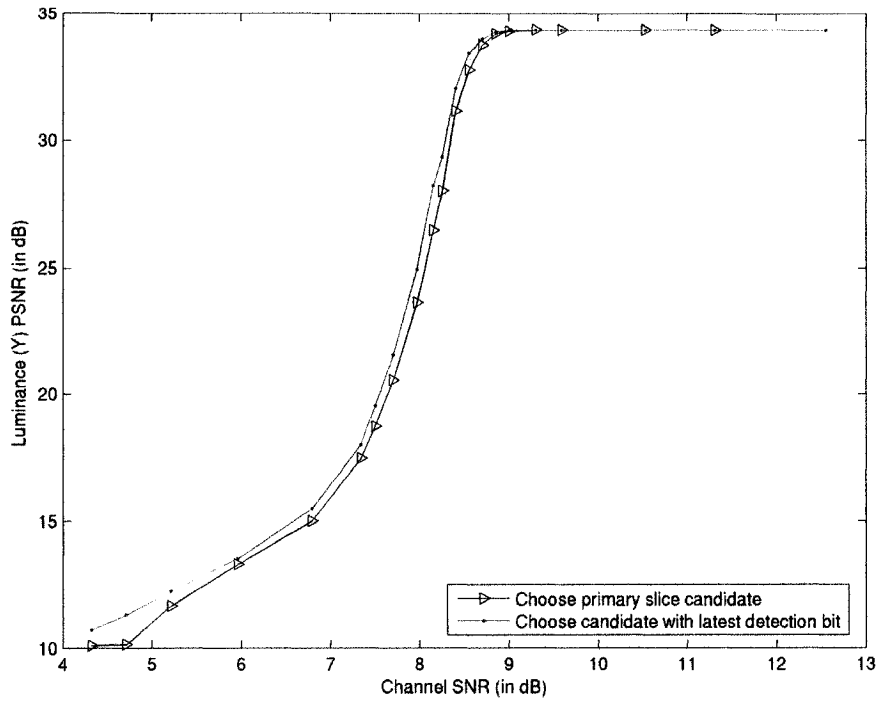

Figure 4.3: Luminance (Y) PSNR vs. channel SNR for video "Table-Tennis" - comparing methods of selecting the winning slice candidate when no slice candidates pass semantic verification

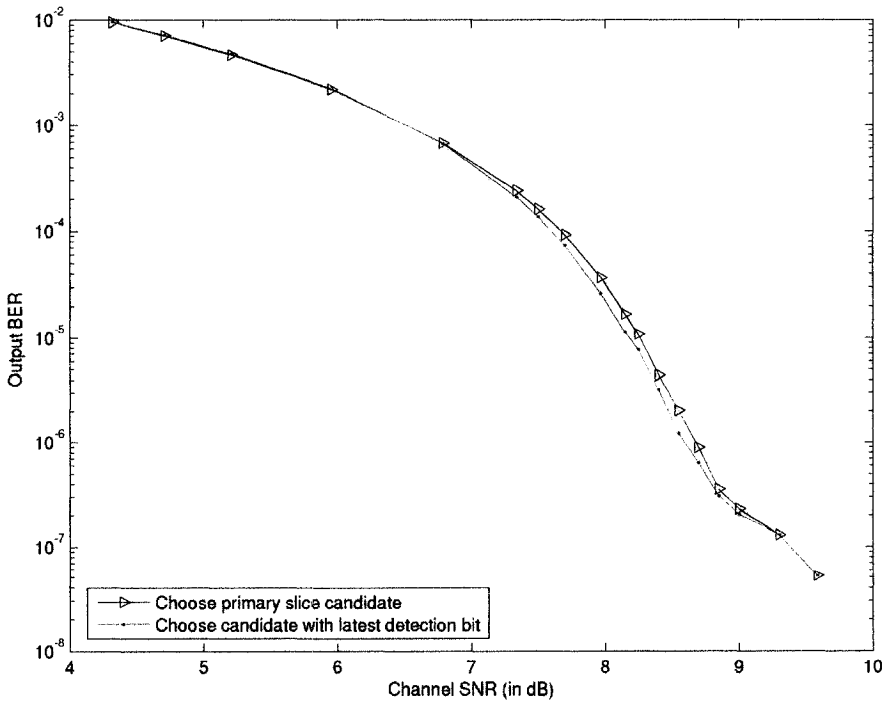

Figure 4.4: BER vs. channel SNR for video "Table-Tennis" - comparing methods of selecting the winning slice candidate when no slice candidates pass semantic verification

Figure 4.3 shows the comparison between the luminance PSNR values of the two methods. The two methods achieve the maximum PSNR at a channel SNR of approximately 9dB or higher. Below 9dB, it is observed that the proposed method yields a slightly higher PSNR than the modified method. Figure 4.4 shows the comparison between the BER values. Between 6.8dB and 9.3dB, it is observed that the proposed method yields a slightly lower BER than the modified method. Outside this range, the two methods result in approximately the same BER.

In summary, the simulation results show that the proposed method uniformly outperforms the modified method. Because the only difference between the two methods is the choice of the winning slice candidate, there is no significant difference between the complexities of the two methods. Therefore, the proposed method is used in the remainder of the simulations.

4.3.2 Objective Performance

The performance of the proposed scheme is measured using PSNR and BER as objective measures. To evaluate the performance of the scheme, two sets of simulations are performed. The first set of simulations is done using the video sequence "Football" and the second set of simulations is done using the video "Table-Tennis". In each set of simulations, the proposed scheme is tested with 50, 100, 300 and 500 slice candidates. The performance of the proposed scheme is compared to the performance of the conventional error concealment scheme described in 4.1.

4.3.2.1 Objective Performance for Video "Football"

The 4-second video sequence "Football" has a frame size of 352x240, and runs at 30 frames per second. The sequence is compressed in H.264 at 1 Mb/s using the Main Profile and CABAC entropy coding. The sequence uses a group of pictures (GOP) of length 15 with IBBPBBP structure.

Without channel noise (i.e. error-free case), the luminance PSNR of the sequence is 28.91dB, which is determined by the compression. When correcting errors, this is the maximum achievable PSNR.

The channel is simulated by inserting noise into the video sequence at a particular channel SNR. The simulation is run at several channel SNR values. At each channel SNR value, the simulation is run 10 times with independent seeds. The result for each channel SNR value is the average of the 10 simulation runs.

Figure 4.5 shows the objective performance comparison in terms of luminance PSNR and Figure 4.6 shows the BER of the output compressed video sequence before it is decompressed. Results are shown for the proposed scheme using 50, 100, 300 and 500 slice candidates, and are compared to the results of the conventional error concealment scheme.

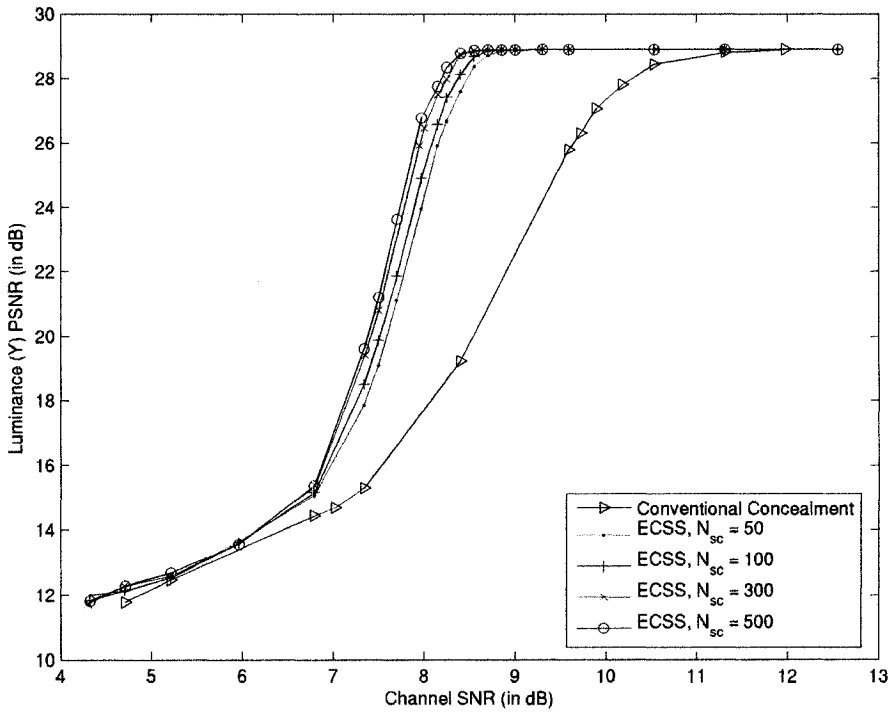

Figure 4.5: Luminance (Y) PSNR vs. channel SNR for video "Football"

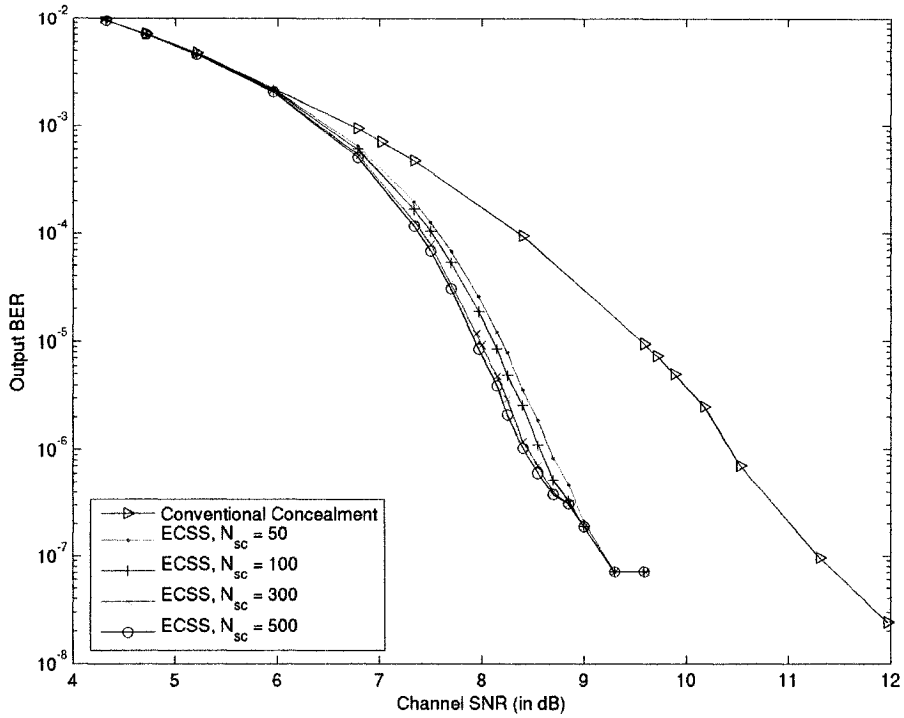

Figure 4.6: Output Bit Error Rate (BER) vs. channel SNR for video "Football"

The minimum channel SNR needed to achieve the maximum PSNR is used to measure the performance gain of the proposed scheme. Table 4.1 compares the minimum channel SNR needed with conventional error concealment and the minimum channel SNR needed with the proposed scheme using 50, 100, 300 and 500 slice candidates.

Scheme	**Channel SNR**
Conventional Concealment	11.95 dB
ECSS – 50 Slice Candidates	8.85 dB
ECSS – 100 Slice Candidates	8.4 dB
ECSS – 300 Slice Candidates	8.25 dB
ECSS – 500 Slice Candidates	8.24 dB

Table 4.1: Minimum Channel SNR needed to achieve maximum luminance (Y) PSNR for video sequence "Football"

Table 4.1 shows that conventional error concealment is capable of achieving the maximum luminance PSNR with a channel SNR of 11.95dB or higher. Table 4.1 also shows that the proposed scheme is capable of achieving the maximum luminance PSNR

at a minimum channel SNR of 8.85dB, 8.4dB and 8.25dB when using 50, 100 and 300 slice candidates respectively. Thus, the proposed scheme provides a gain in channel SNR of 3.1dB, 3.55dB and 3.7dB when using 50, 100 and 300 slice candidates respectively, as seen in Table 4.2.

Scheme	Gain
ECSS – 50 Slice Candidates	3.1 dB
ECSS – 100 Slice Candidates	3.55 dB
ECSS – 300 Slice Candidates	3.70 dB
ECSS – 500 Slice Candidates	3.7 dB

Table 4.2: Performance gain of the proposed scheme in comparison to conventional error concealment for video sequence "Football"

The results also show that using 500 slice candidates yields a marginal gain over using 300 slice candidates. The maximum luminance PSNR when using 500 slice candidates is achieved at a channel SNR slightly below 8.25dB. This results in a gain over conventional error concealment slightly above 3.7dB.

When comparing the transition regions of the curves in Figure 4.5, the slopes for all the curves for the proposed scheme have approximately the same slope, while the curve for the conventional error concealment scheme has a gentler slope. The similarity of the slopes for the proposed scheme may be due to the fact that, when a certain channel SNR is reached, there are too many errors in a given slice for the SSV to be able to correct. As a result, the PSNR of the slice drops significantly at that point. In contrast, semantic errors are never corrected in the conventional error concealment scheme. They will always cause degradation in video quality and hence, the PSNR declines less suddenly.

Little to no gain occurs at the bottom end of the curve. This could be due to the fact that it is difficult for the SSV to yield any improvements when there are too many errors in each slice. Since the SSV chooses the slice candidate with the latest detection bit, it is possible that the winning slice candidate decreases the PSNR of a slice with a large number of errors in it. Regardless, the quality of the video in this region is extremely poor. Even a minor gain would not make it usable.

It was also observed that the chrominance PSNR values (not shown) drop off in a similar manner to the luminance PSNR values.

4.3.2.2 *Objective Performance for Video "Table-Tennis"*

The video sequence "Table-Tennis" has the same frame size and length as the video sequence "Football" and is compressed using the same parameters. Like with "Football", simulations were performed with 10 independent seeds at several channel SNR values.

The maximum luminance PSNR of the sequence is 34.34dB. This is higher than the maximum luminance PSNR for "Football", which was compressed under the same conditions. This could be because there is less motion in "Table-Tennis", which usually results in less residual data for each macroblock.

Figure 4.7 shows the performance comparison for luminance PSNR and Figure 4.8 shows the performance comparison for BER. The chrominance PSNR curves are similar to the luminance PSNR curve. Results are shown for the proposed scheme using the same number of slice candidates as in "Football" and are again compared to the results for conventional error concealment.

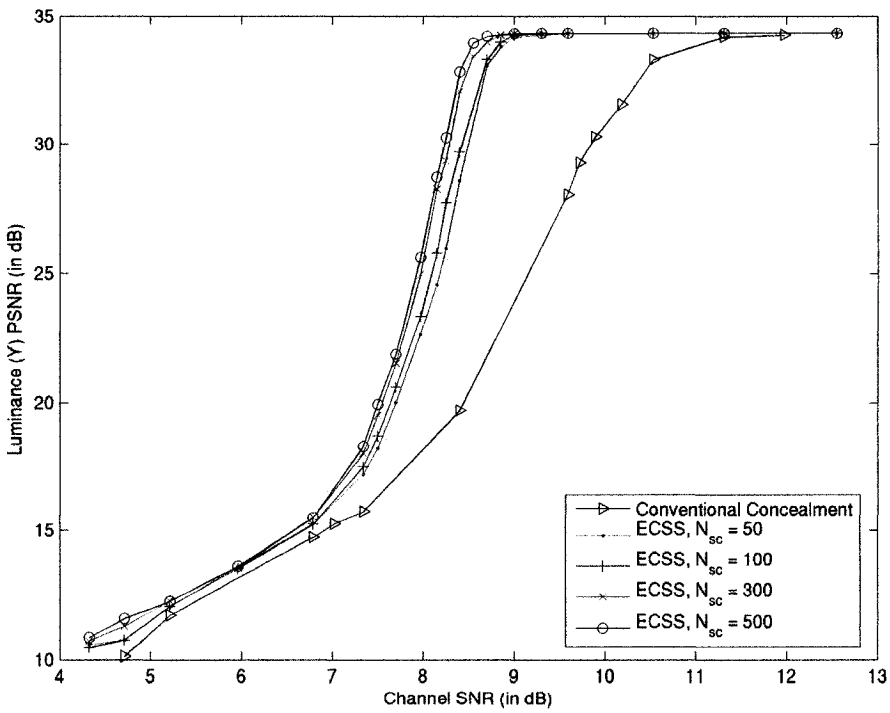

Figure 4.7: Luminance (Y) PSNR vs. channel SNR for video "Table-Tennis"

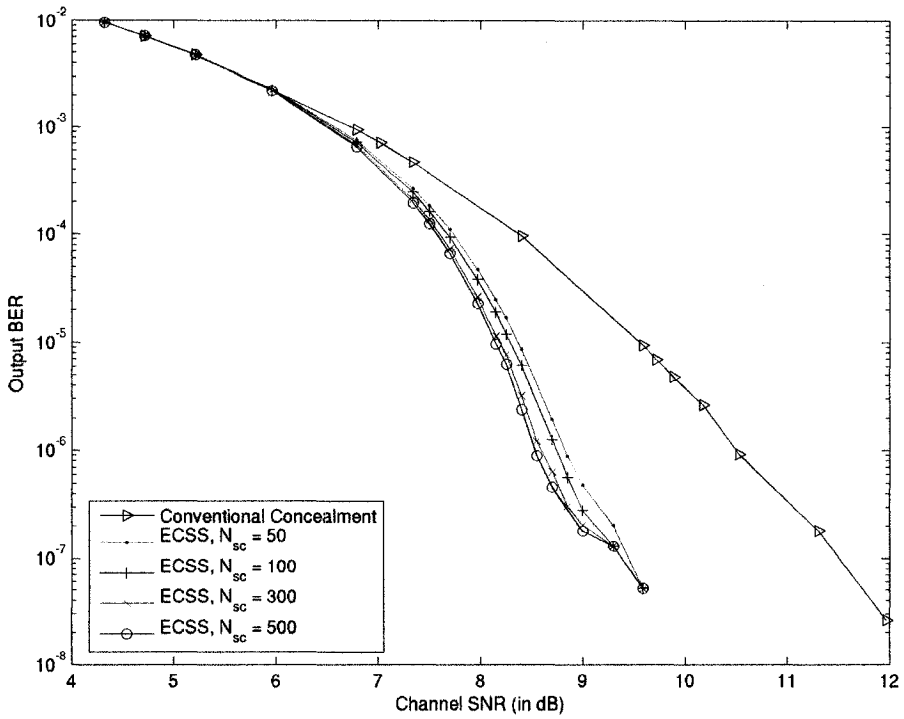

Figure 4.8: Output Bit Error Rate (BER) vs. channel SNR for video "Table-Tennis"

Table 4.3 shows the minimum channel SNR needed to achieve the maximum luminance PSNR for conventional error concealment and for the proposed scheme. Table 4.4 shows the resulting gains in channel SNR the proposed scheme provides (for each number of slice candidates tested) over conventional concealment. With the exception of the scheme with 50 slice candidates, the gains in channel SNR are slightly lower than the gains seen for "Football". The gains are, however, still significant.

Scheme	**Channel SNR**
Conventional Concealment	12.05 dB
ECSS – 50 Slice Candidates	9.00 dB
ECSS – 100 Slice Candidates	8.95 dB
ECSS – 300 Slice Candidates	8.85 dB
ECSS – 500 Slice Candidates	8.7 dB

Table 4.3: Minimum Channel SNR needed to achieve maximum luminance (Y) PSNR for video sequence "Table-Tennis"

Scheme	Gain
ECSS – 50 Slice Candidates	3.05 dB
ECSS – 100 Slice Candidates	3.10 dB
ECSS – 300 Slice Candidates	3.20 dB
ECSS – 500 Slice Candidates	3.35 dB

Table 4.4: Performance gain of the proposed scheme in comparison to conventional error concealment for video sequence "Table-Tennis"

4.3.3 Subjective Performance

The subjective performance of the proposed scheme is evaluated by looking at decompressed video sequences. The same two video sequences ("Football" and "Table-Tennis") that were used in the objective performance evaluation are used in the subjective performance evaluation.

To evaluate subjective performance, the output video of the proposed scheme was compared to the output video of the conventional concealment scheme. The output of proposed scheme was observed using 100, 300 and 500 slice candidates. This section presents comments based on the subjective observations. Also, several of the decompressed frames are presented for the reader's own assessment.

4.3.3.1 Subjective Performance for Video "Football"

When looking at the decompressed video "Football", there are obvious improvements in picture quality when the proposed scheme is used in comparison to conventional error concealment. There are further improvements in picture quality as the number of slice candidates in the proposed scheme is increased. At 8dB channel SNR, the decompressed video using conventional error concealment is blocky and a significant portion of each frame is lost. In contrast, when the proposed scheme using 100 slice candidates is employed at 8dB channel SNR, the sequence is mostly viewable, although several frames have visible errors. When the number of slice candidates is increased to 300, the picture quality is improved. The number of visible errors decreases by about one half. When the number of slice candidates is further increased to 500, the picture quality is further improved, however this improvement is less significant than the improvement seen between 100 and 300 slice candidates.

For the reader's own assessment, frames 58 and 122 are presented after having been transmitted over an AWGN at 8dB and after having been decompressed using conventional error concealment as well as the proposed scheme. Figure 4.9 to Figure 4.13 show the results for frame 58 using each of the schemes. Figure 4.14 to Figure 4.18 show the results for frame 122 using each of the schemes. The luminance PSNR values of frames 58 and 122 that correspond to each of the decoding schemes are presented in Table 4.5 and Table 4.6 respectively. The luminance PSNR values of the error-free case are included in each table to show the maximum achievable PSNR for the frame. The decompressed frames and the PSNR values both show that performance is improved by the proposed scheme and that the improvements increase as the number of slice candidates increases.

Scheme	PSNR (Y)
Error-free frame	28.77 dB
Conventional Concealment	15.92 dB
ECSS – 100 Slice Candidates	26.59 dB
ECSS – 300 Slice Candidates	27.95 dB
ECSS – 500 Slice Candidates	28.08 dB

Table 4.5: Luminance PSNR of decompressed frame 58 of video sequence "Football" transmitted over AWGN channel at 8dB channel SNR

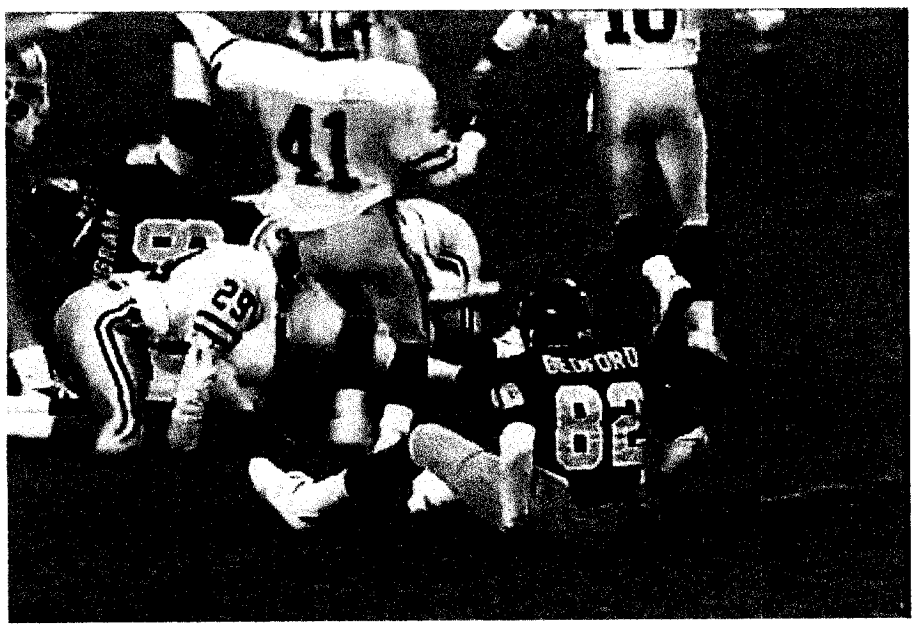

Figure 4.9: Decompressed error-free frame 58 of video sequence "Football"

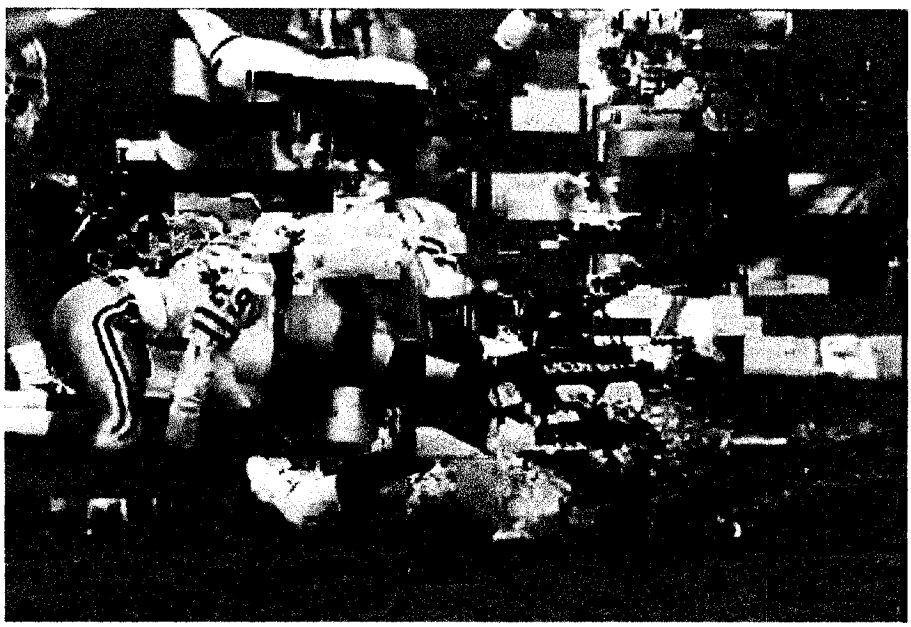

Figure 4.10: Frame 58 of video sequence "Football" transmitted over AWGN channel at 8dB SNR and decoded using conventional error concealment

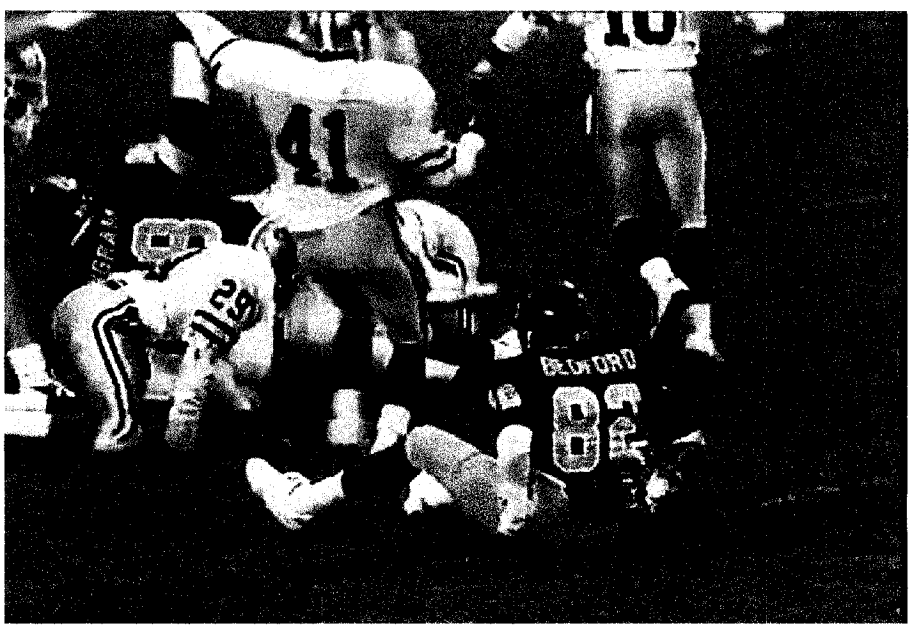

Figure 4.11: Frame 58 of video sequence "Football" transmitted over AWGN channel at 8dB SNR and decoded using ECSS with 100 slice candidates

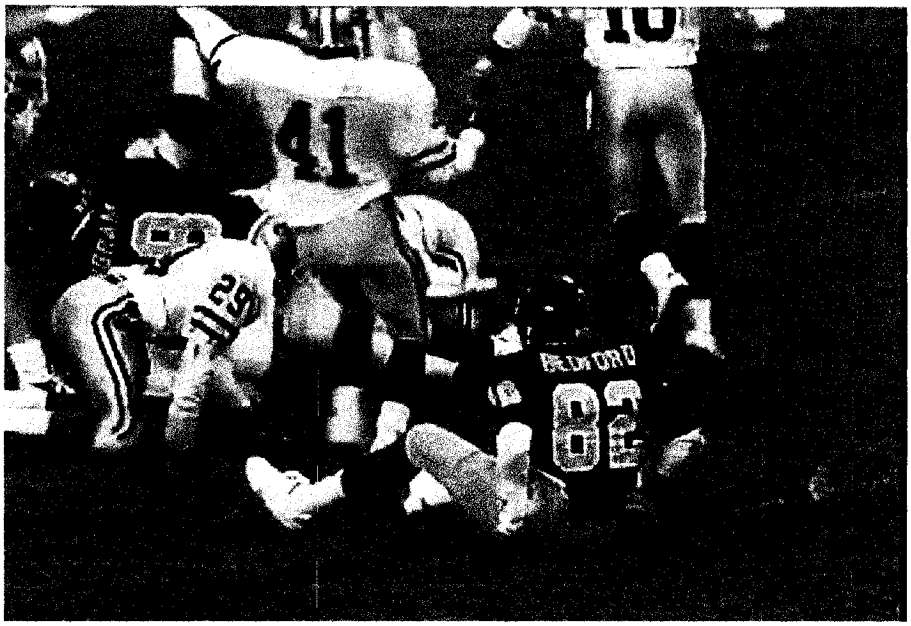

Figure 4.12: Frame 58 of video sequence "Football" transmitted over AWGN channel at 8dB SNR and decoded using ECSS with 300 slice candidates

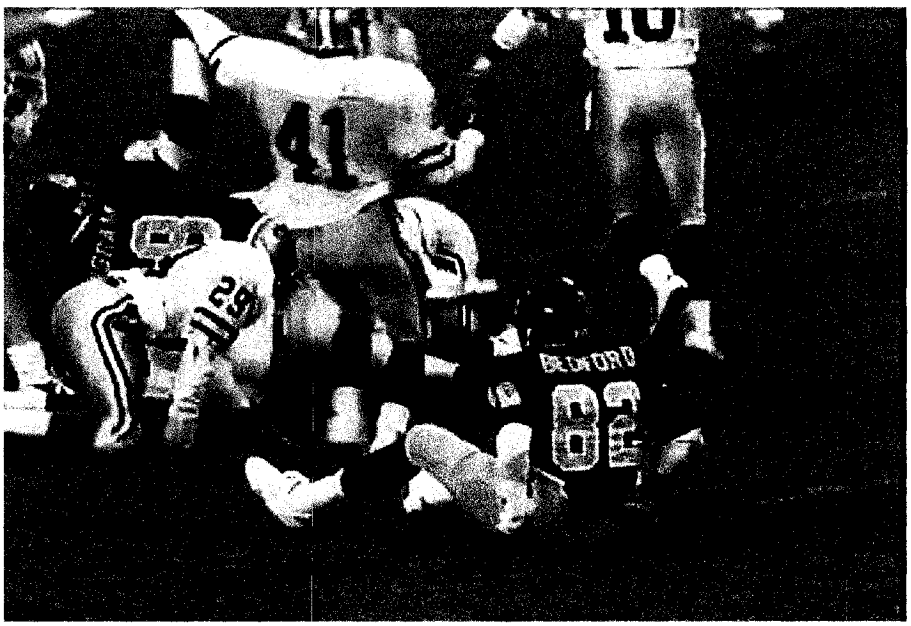

Figure 4.13: Frame 58 of video sequence "Football" transmitted over AWGN channel at 8dB SNR and decoded using ECSS with 500 slice candidates

Scheme	PSNR (Y)
Error-free frame	29.47 dB
Conventional Concealment	16.17 dB
ECSS – 100 Slice Candidates	23.64 dB
ECSS – 300 Slice Candidates	25.10 dB
ECSS – 500 Slice Candidates	28.57 dB

Table 4.6: Luminance PSNR of decompressed frame 122 of video sequence "Football" transmitted over AWGN channel at 8dB channel SNR

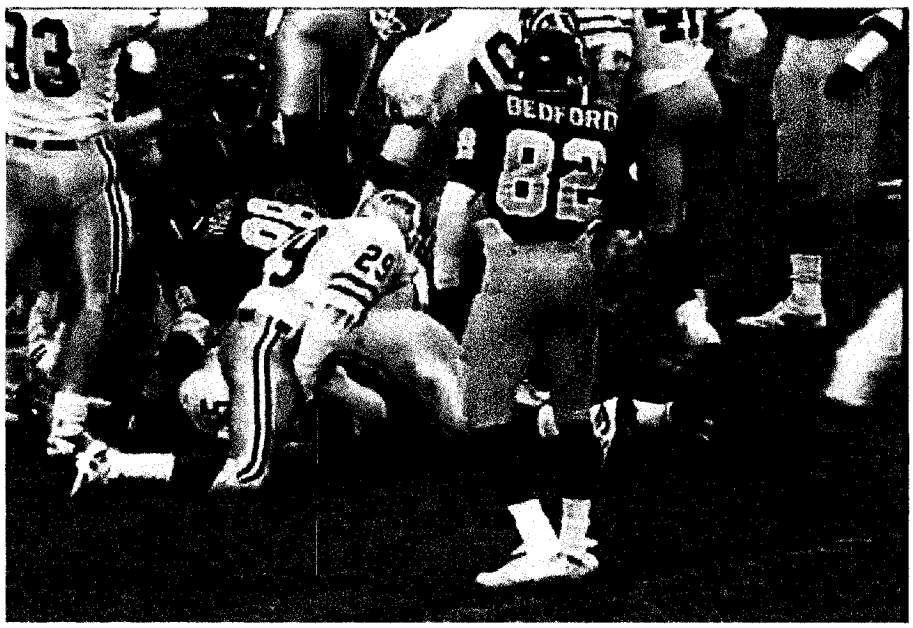

Figure 4.14: Decompressed error-free frame 122 of video sequence "Football"

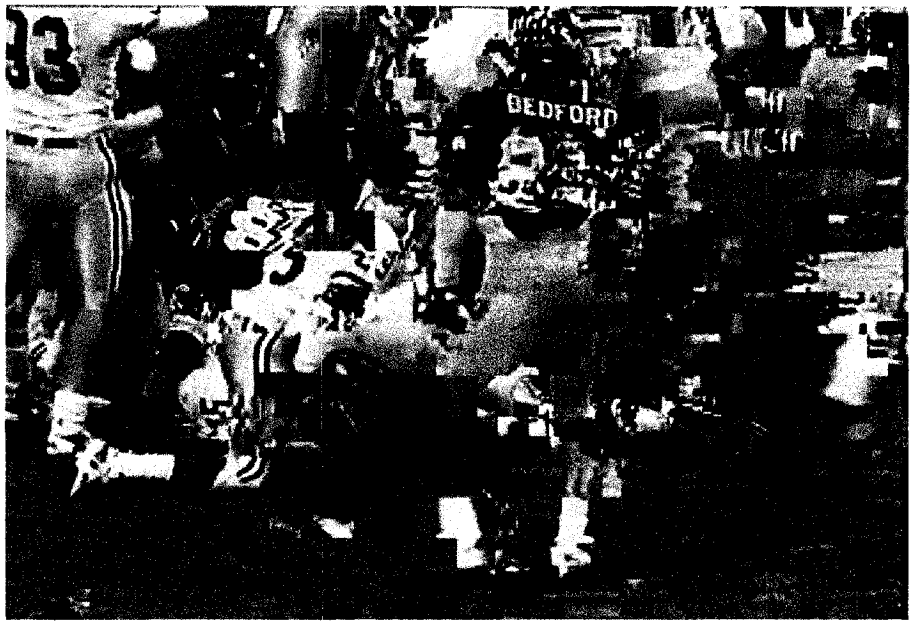

Figure 4.15: Frame 122 of video sequence "Football" transmitted over AWGN channel at 8dB SNR and decoded using conventional error concealment

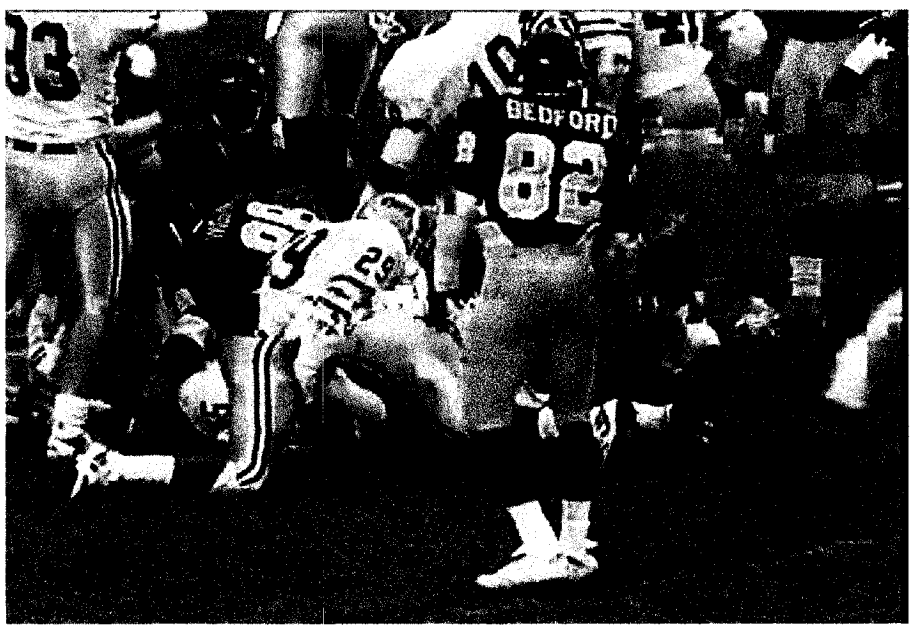

Figure 4.16: Frame 122 of video sequence "Football" transmitted over AWGN channel at 8dB SNR and decoded using ECSS with 100 slice candidates

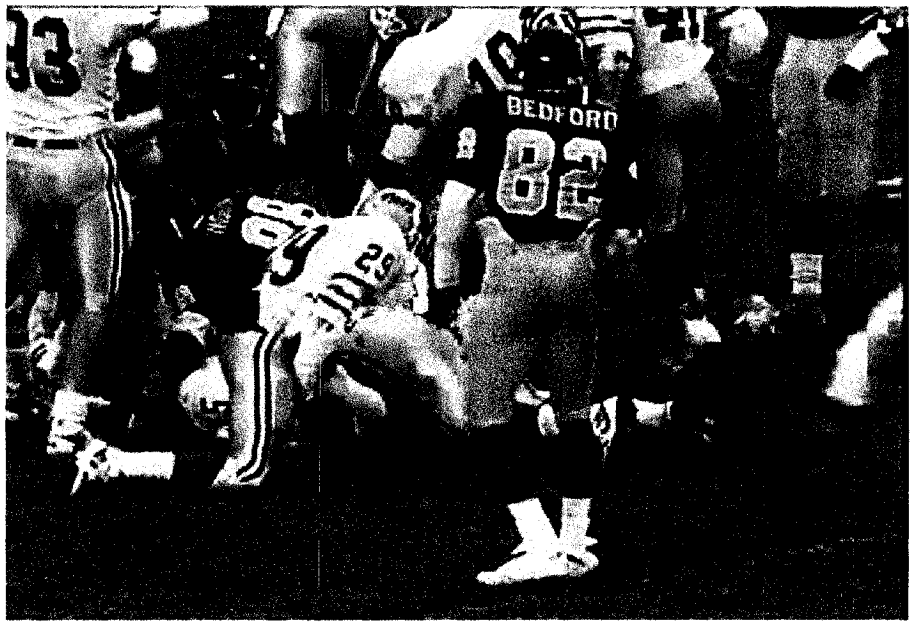

Figure 4.17: Frame 122 of video sequence "Football" transmitted over AWGN channel at 8dB SNR and decoded using ECSS with 300 slice candidates

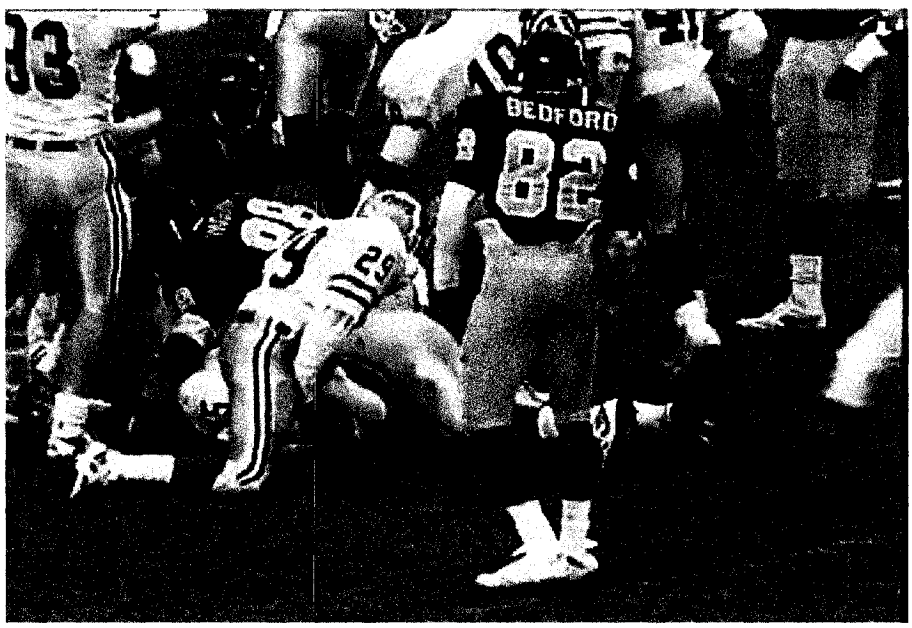

Figure 4.18: Frame 122 of video sequence "Football" transmitted over AWGN channel at 8dB SNR and decoded using ECSS with 500 slice candidates

4.3.3.2 Subjective Performance for Video "Table-Tennis"

When using the video sequence "Table-Tennis", simulations are run at 8.3dB channel SNR. As in "Football", there are obvious improvements in picture quality when the proposed scheme is used in comparison to conventional error concealment. Once again, there are further improvements in picture quality as the number of slice candidates in the proposed scheme is increased.

For the reader's own assessment, frames 59 and 89 are presented after having been transmitted over an AWGN at 8.3dB and after having been decompressed using conventional error concealment as well as the proposed scheme. Figure 4.19 to Figure 4.23 show the results for frame 59 using each of the schemes. Figure 4.24 to Figure 4.28 show the results for frame 89 using each of the schemes. The corresponding component PSNR values for each of the decoding schemes are presented in Table 4.7 and Table 4.8 respectively.

As was observed for "Football", the output frames and the PSNR values both show that performance is improved by the proposed scheme, and that the improvements mostly increase as the number of slice candidates increases.

Scheme	**PSNR (Y)**
Error-free frame	33.78 dB
Conventional Concealment	20.80 dB
ECSS – 100 Slice Candidates	24.42 dB
ECSS – 300 Slice Candidates	26.74 dB
ECSS – 500 Slice Candidates	33.78 dB

Table 4.7: Luminance PSNR of decompressed frame 59 of video sequence "Table-Tennis" transmitted over AWGN channel at 8.3dB channel SNR

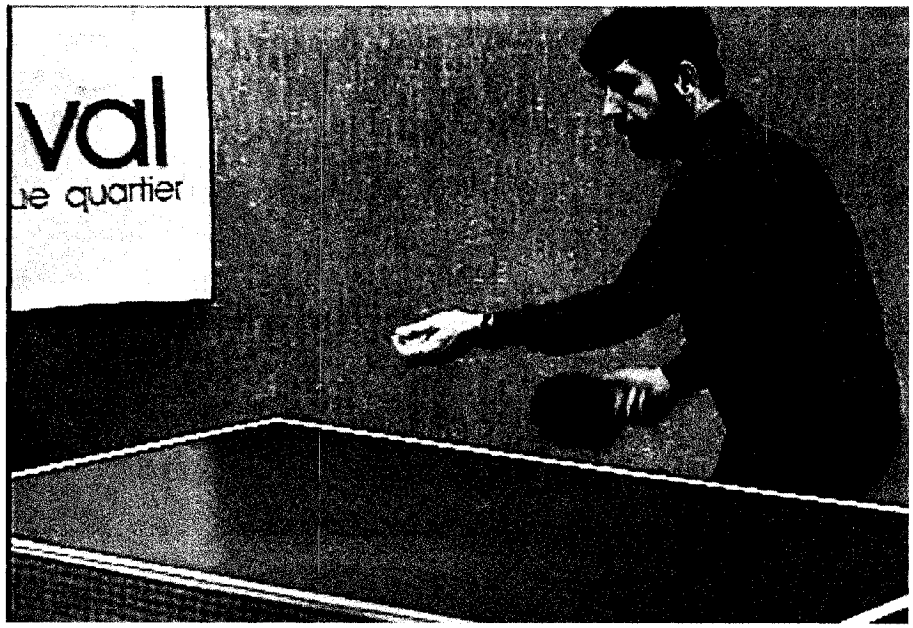

Figure 4.19: Decompressed error-free frame 59 of video sequence "Table-Tennis"

Figure 4.20: Frame 59 of video sequence "Table-Tennis" transmitted over AWGN channel at 8.3dB SNR and decoded using conventional error concealment

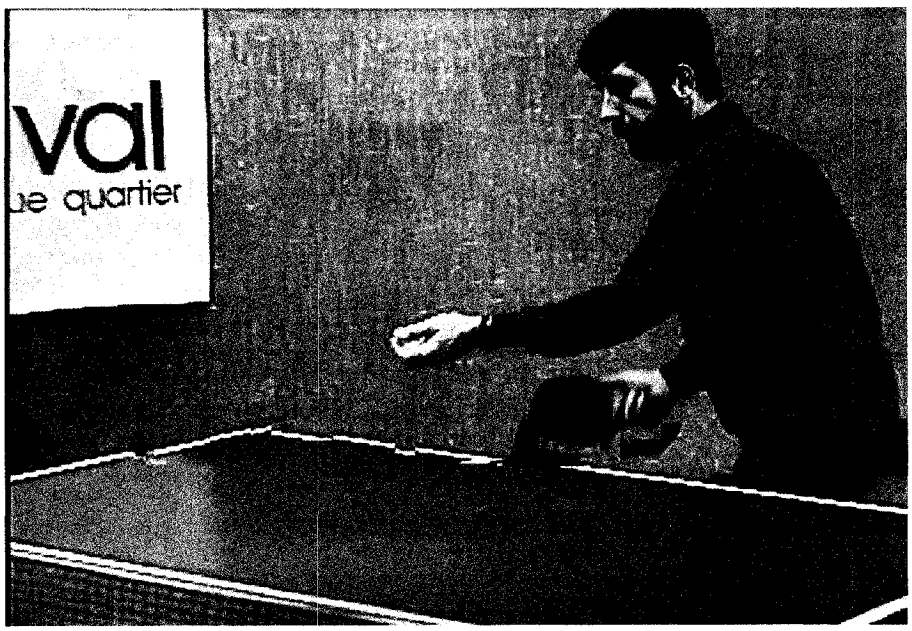

Figure 4.21: Frame 59 of video sequence "Table-Tennis" transmitted over AWGN channel at 8.3dB SNR and decoded using ECSS with 100 slice candidates

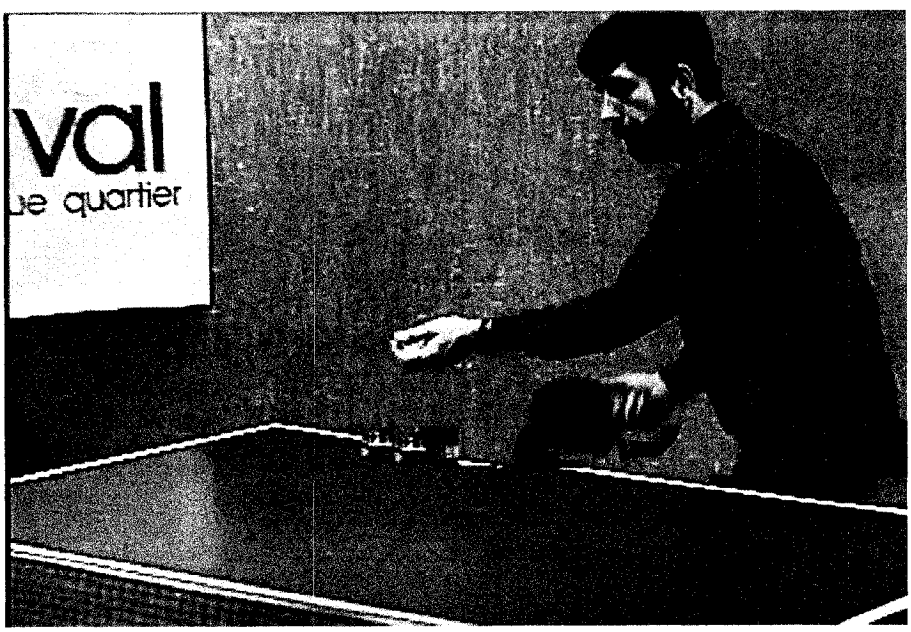

Figure 4.22: Frame 59 of video sequence "Table-Tennis" transmitted over AWGN channel at 8.3dB SNR and decoded using ECSS with 300 slice candidates

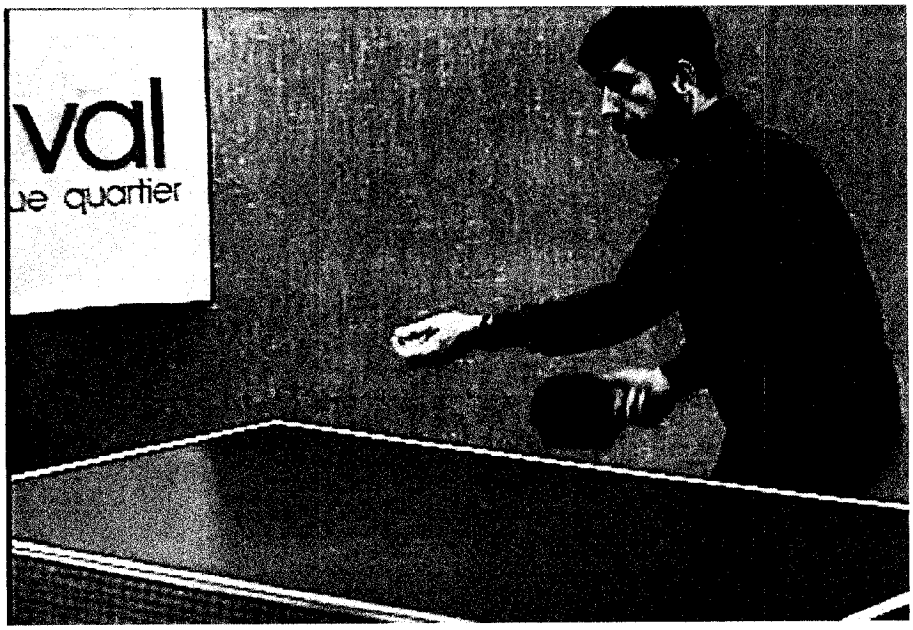

Figure 4.23: Frame 59 of video sequence "Table-Tennis" transmitted over AWGN channel at 8.3dB SNR and decoded using ECSS with 500 slice candidates

Scheme	PSNR (Y)
Error-free frame	37.92 dB
Conventional Concealment	19.60 dB
ECSS – 100 Slice Candidates	26.34 dB
ECSS – 300 Slice Candidates	27.15 dB
ECSS – 500 Slice Candidates	35.98 dB

Table 4.8: Luminance PSNR of decompressed frame 89 of video sequence "Table-Tennis" transmitted over AWGN channel at 8.3dB channel SNR

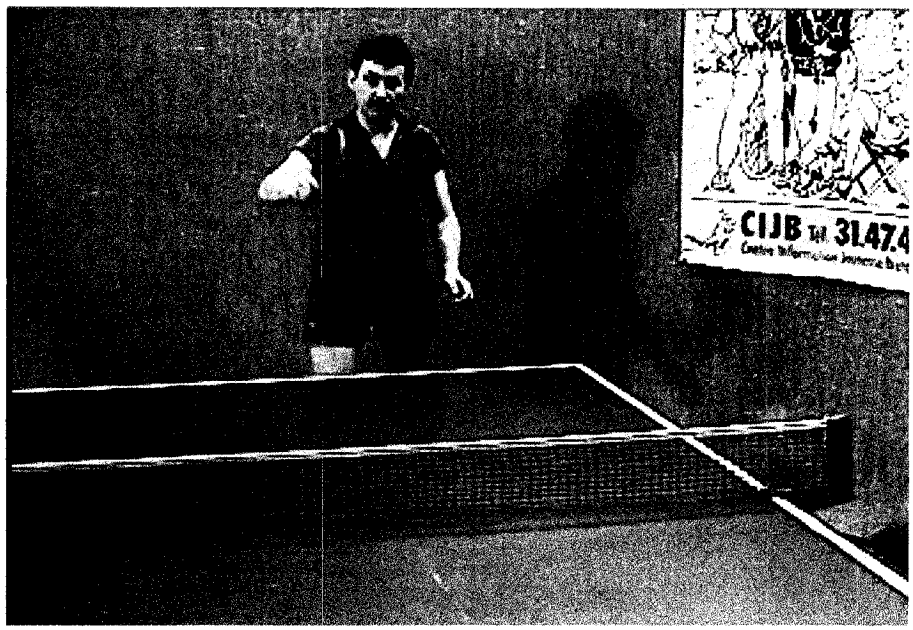

Figure 4.24: Decompressed error-free frame 89 of video sequence "Table-Tennis"

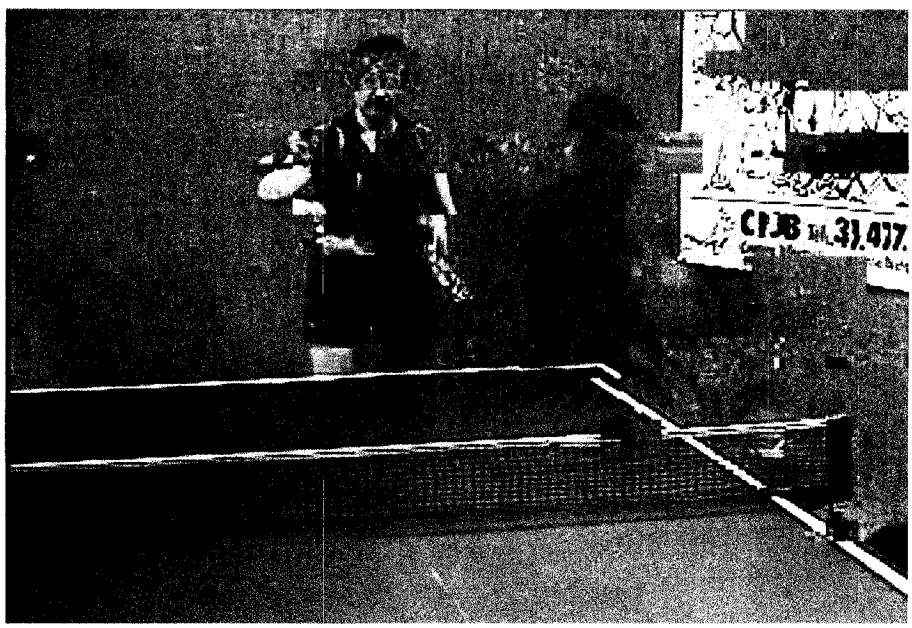

Figure 4.25: Frame 89 of video sequence "Table-Tennis" transmitted over AWGN channel at 8.3dB SNR and decoded using conventional error concealment

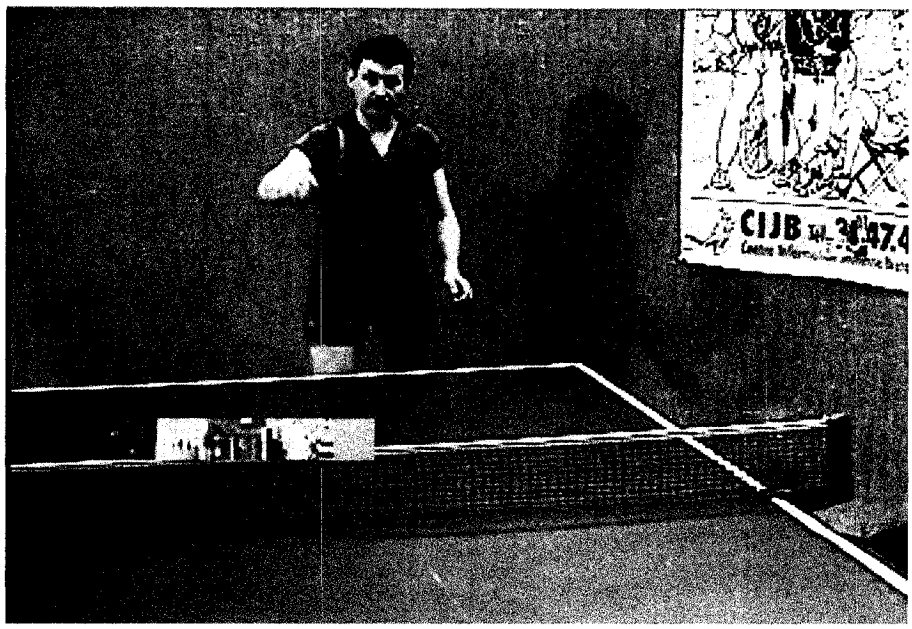

Figure 4.26: Frame 89 of video sequence "Table-Tennis" transmitted over AWGN channel at 8.3dB SNR and decoded using ECSS with 100 slice candidates

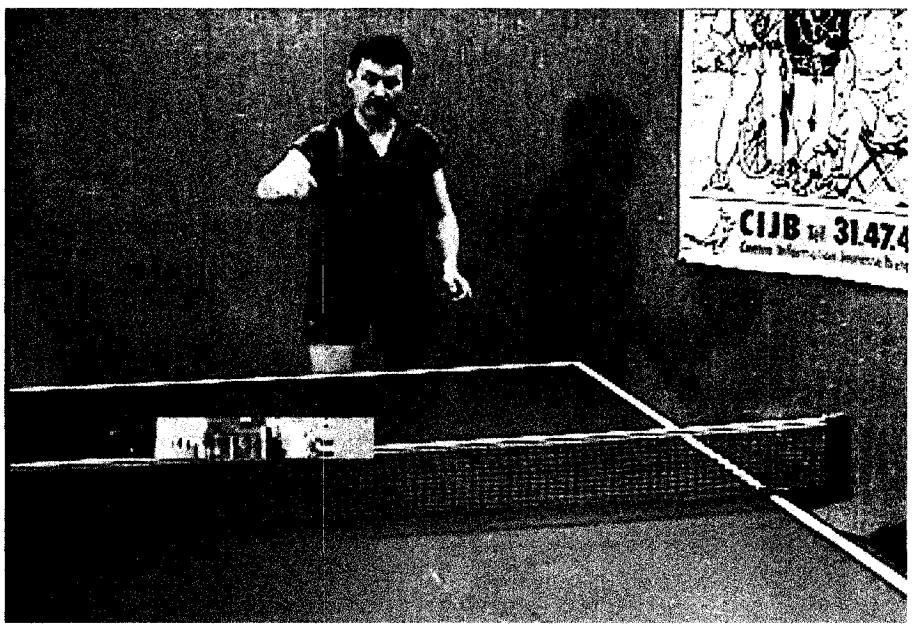

Figure 4.27: Frame 89 of video sequence "Table-Tennis" transmitted over AWGN channel at 8.3dB SNR and decoded using ECSS with 300 slice candidates

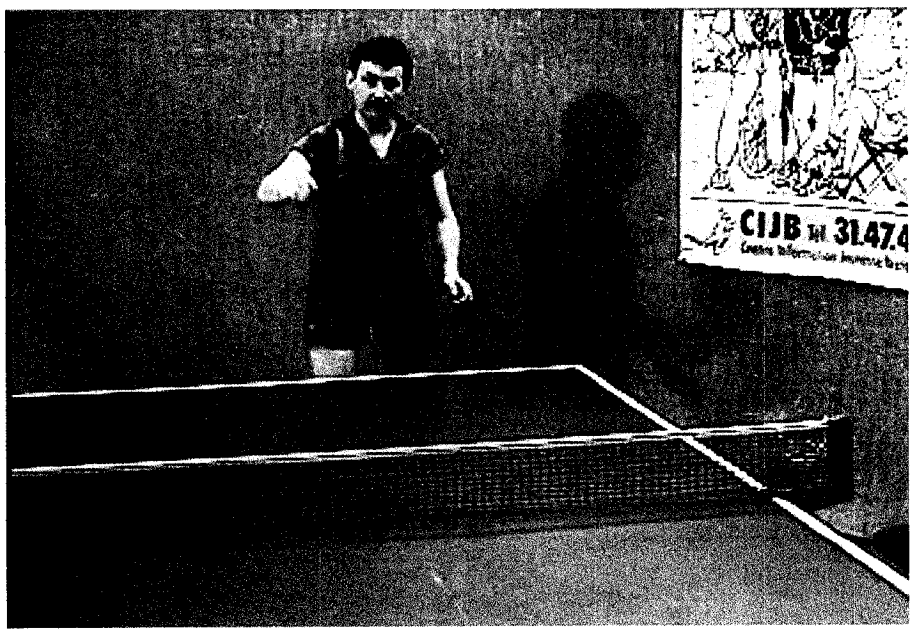

Figure 4.28: Frame 89 of video sequence "Table-Tennis" transmitted over AWGN channel at 8.3dB SNR and decoded using ECSS with 500 slice candidates

4.3.4 Complexity of the Proposed Scheme

The complexity of the proposed scheme is evaluated by comparing it to the complexity of the conventional error concealment scheme. A set of simulations was performed to evaluate the complexity. The simulation conditions are identical to the simulation conditions for the objective and subjective performance evaluation.

All of the components of the conventional error concealment scheme are part of the proposed scheme. In addition to the common components, the proposed scheme also includes the SCG and the SSV.

As discussed in Section 4.2.2, the complexity of the SCG is $n_{sc}\log(N)$ where n_{sc} is the number of slice candidates generated and N is the length of the slice.

Unlike the complexity of the SCG, the complexity of the SSV is difficult to quantify because the SSV is a complicated system (when reading a slice candidate, the SSV partially decompresses it). Previous schemes addressed this issue by counting the number of extracted bits and the number of extracted slices, and using them as complexity measures [35],[36],[39]. In this thesis, those measures are used as well as two others: the number of extracted macroblocks and the time ratio. The time ratio compares the run

time of a module of or the entire scheme to the run time of the H.264 decompressor. The simulations in this thesis are run on a 1.67GHz PowerPC G4 processor.

4.3.4.1 Complexity of the SSV

In Section 4.2.3, it was discussed that the time it takes the SSV to check a slice candidate was reduced by restarting at the beginning of the macroblock that contains the first flip-bit rather than restarting at the beginning of the slice. To illustrate the amount of time saved, a simulation was done to compare the two methods of starting a slice candidate. The simulation employed the proposed scheme with 300 slice candidates to decode the video sequence "Football". The time ratio measure for this simulation is shown in Figure 4.29. It can be observed that below 8.5dB channel SNR, the method used in the proposed scheme offers a savings in time of approximately 40%.

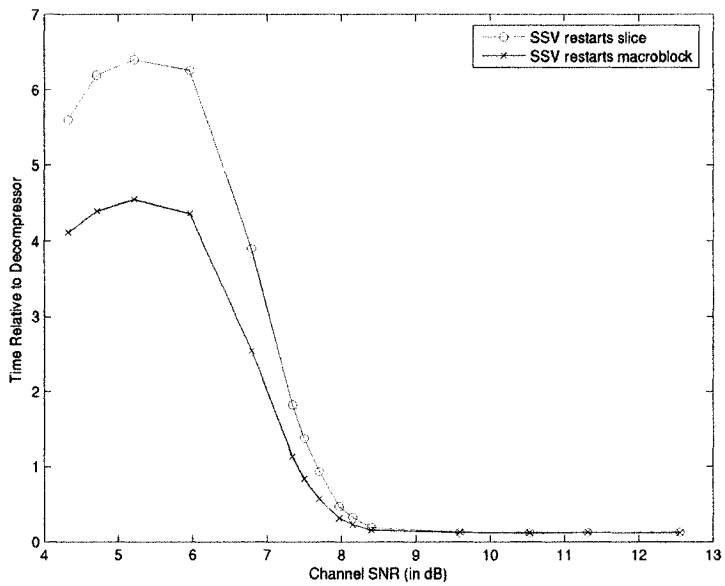

Figure 4.29: Time ratio between the SSV and the H.264 decompressor for video sequence "Football". Proposed scheme uses 300 slice candidates.

The complexity of the SSV is now measured using each of the four measures described. The video sequence "Table-Tennis" is used in the simulation. Results are shown for the proposed scheme using 50, 100, 200, 300 and 500 slice candidates. Each of the complexity measures is presented as a function of channel SNR and the results are shown from Figure 4.30 to Figure 4.33.

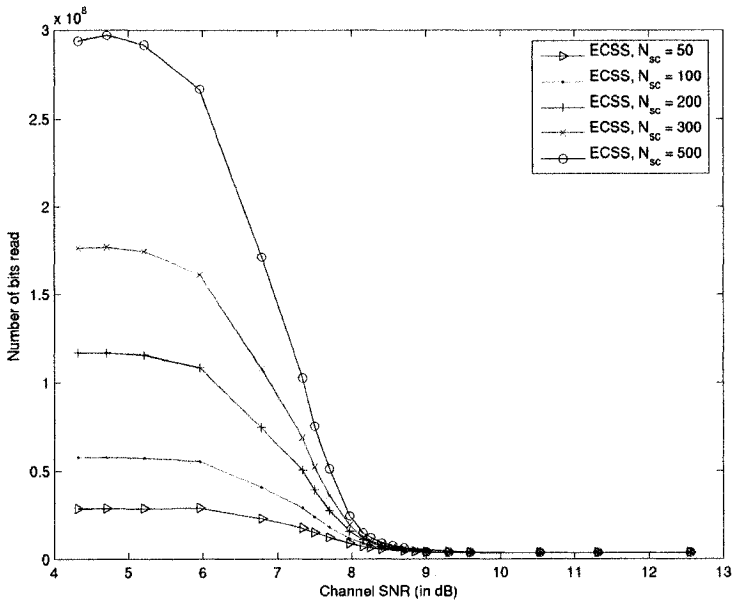

Figure 4.30: Number of bits extracted by the SSV for video sequence "Table-Tennis"

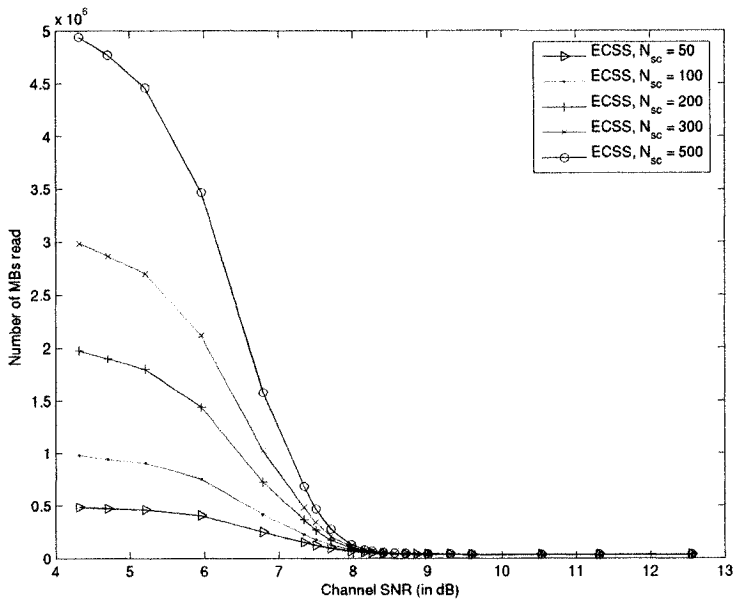

Figure 4.31: Number of macroblocks extracted by the SSV for video sequence "Table-Tennis"

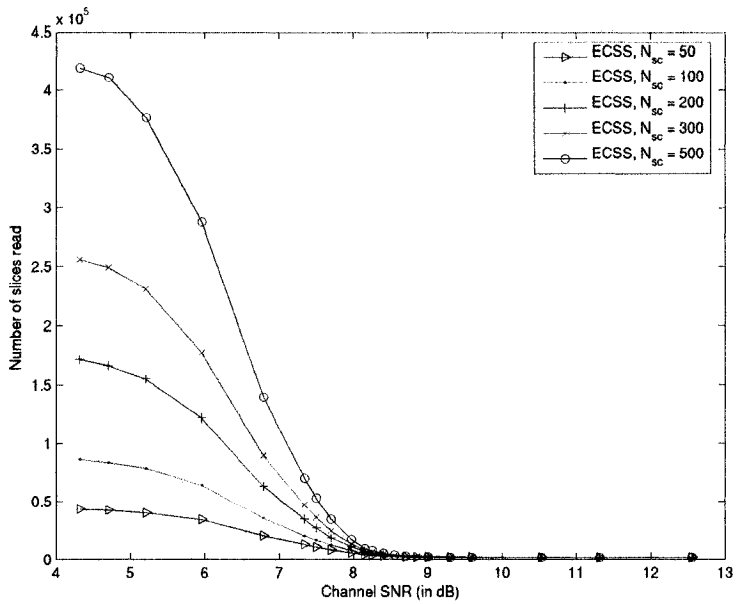

Figure 4.32: Number of slice candidates read by the SSV for video sequence "Table-Tennis"

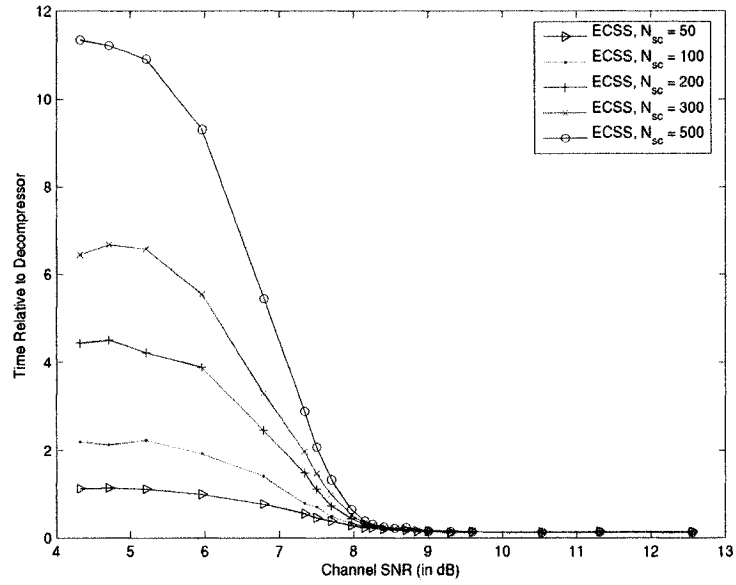

Figure 4.33: Time ratio between the SSV and the H.264 decompressor for video sequence "Table-Tennis"

When comparing the complexity measures between the different numbers of slice candidates generated, the four measures give similar results. When the channel SNR is above 8.5 dB, the SSV has approximately the same complexity regardless of the number of slice candidates used. This is because the SSV finds a winning slice candidate quickly

and does not have to check many candidates. Below 8.5dB, when the proposed scheme uses more slice candidates, each of its corresponding complexity measures is higher. Based on the curves, the relationship between number of slice candidates and complexity of the SSV appears to be approximately linear for all channel SNR values.

When the channel SNR is below 4.7dB, reducing the channel SNR sometimes decreases the complexity. When there are many errors in a slice, it is likely that a semantic error will be detected early. The earlier the detection bit occurs, the greater the number of slice candidates that have no flip-bits before the detection bit. The SSV skips any slice candidate that has no flip-bits before the detection bit. This seems to be the reason for the decrease in complexity.

When setting the number of slice candidates to generate, a tradeoff is made between the complexity of the SSV and performance gain. When looking at the PSNR curves in Section 4.3.2, the performance gain between curves for the proposed scheme is noticeable up to 300 slice candidates. Between 300 and 500 slice candidates, the performance gain is very small while the increase in complexity is large. Based on this observation, it is recommended to generate 300 slice candidates, although this may vary depending on complexity requirements.

4.4 Implementation Considerations

A practical implementation of a decoding scheme for compressed video should be able to run in real time, for either a streaming or interactive application. In order to for a scheme to achieve real time, its run time must be reduced. Two ways to reduce run time are parallelization and pipelining. Parallelization consists of running several tasks simultaneously instead of sequentially. Pipelining involves having each module take in new data as soon as it has output its current data to the next module. There are several parts of ECSS that can either be parallelized or pipelined.

All slices in a frame can be processed in parallel. This can be done because decompressing a slice never requires previously decompressed data from other slices in the same frame. Thus, a separate SCG, SSV and H.264 Decompressor would be implemented for each slice. In the experiments performed in this thesis, each frame has 15 slices. Thus the run time of the scheme can be reduced by about a factor of 15.

Further parallelization can be done, as many slice candidates can be verified simultaneously. Rather than having a single SSV that goes through the slice candidates one by one, each slice candidate could be checked by a separate SSV. The scheme would have to be changed, as each SSV would only verify one candidate. An additional module would compare the results from each SSV and select the appropriate winning slice candidate. Verifying slice candidates in parallel would ensure that the SSV run time would be as low as possible and would not depend on channel SNR.

The modules in ECSS can also be pipelined. The SCG, SSV and H.264 Decompressor can be designed such that each module takes in its next input as soon as it has passed its output to the next module. Thus, the time it takes to decompress a frame can be used to prepare the next frame for decompression.

After parallelization and pipelining, the two main bottlenecks of the ECSS decoder would be the H.264 Decompressor and the SCG. The run time of the H.264 Decompressor is beyond the scope of this discussion. The SCG uses the IPSA to generate and rank slice candidates. The IPSA can neither be parallelized nor pipelined. The main factor that determines the run time of the algorithm is the number of slice candidates generated. Since the performance of ECSS increases as the number of slice candidates increases, there is a tradeoff between performance and run time.

4.5 Summary

This chapter presented a transmission scheme for H.264 compressed video in which no channel code is present. The proposed scheme, ECSS, uses channel information to aid source decoding. The absolute values of the received soft values are used to generate a finite list of slice candidates and rank them in descending order of likelihood. The first slice candidate that passes semantic verification is chosen as the winning candidate, because it is the most likely slice candidate to pass semantic verification. If all slice candidates fail semantic verification, the winning slice candidate is the one with the latest detection bit.

ECSS is a non-iterative decoding scheme that does not depend on a channel code. It uses channel information to aid source decoding, but cannot use source information to aid channel decoding. The scheme could, be adapted to include a channel code. Any channel

code that produces soft-output bits could be used to deliver the soft-values needed to aid source decoding. In the next chapter, a decoding scheme will be introduced that uses a soft-output channel code. The scheme uses the method presented in this chapter to feed channel information into the source decoder and presents a method to feed source information back into the channel decoder. Thus information is iteratively exchanged between the source and channel decoders.

Chapter 5

Iterative Joint Source-Channel Decoding of H.264

Chapter 4 presented ECSS, a decoding scheme for H.264 compressed video without a channel code. ECSS improves performance by using H.264 source semantics to conceal errors. The soft-bit values received from the channel are used to select and rank a set of slice candidates in descending order of likelihood and a winner is chosen among the candidates.

This chapter proposes a scheme that builds on ECSS and adds a channel code. The channel code used is a convolutional code and soft decoding is used to provide soft values to the source decoder. This allows the scheme to add an iterative element to the decoding of H.264 video. A method of feeding source information back into the channel decoder is thus proposed. A conference paper [42] was published from the contents of this chapter.

In this chapter, Section 5.1 presents the proposed scheme, Iterative Joint Source-Channel Decoding (IJSCD), and gives a detailed description of each of its modules. The performance of the scheme is evaluated both objectively and subjectively in Section 5.2. Section 5.2 also analyzes the complexity of the proposed IJSCD scheme. Section 5.3 discusses design considerations for a hardware implementation of the scheme. The chapter concludes with a summary in Section 5.4.

5.1 Iterative Joint Source-Channel Decoding (IJSCD)

The proposed scheme, Iterative Joint Source-Channel Decoding (IJSCD), is a candidate-based joint source-channel decoding scheme that builds on ECSS from the previous chapter and follows a similar framework to several of the schemes discussed in Section 2.3.

IJSCD most closely resembles IJSCDTC [39] in that source information is used to modify channel information (and vice versa) in an iterative manner. There are however, significant differences between IJSCDTC and the proposed scheme. First, while IJSCDTC uses conventional turbo coding with two convolutional codes, the proposed scheme uses a single convolutional code. Thus, the turbo dynamic is between the source decoder and the channel decoder. Second, the MPEG-4 used in IJSCDTC uses VLC for entropy compression. In contrast, the proposed scheme uses CABAC for entropy coding.

The block diagram of the proposed scheme, Iterative Joint Source-Channel Decoding, is shown in Figure 5.1.

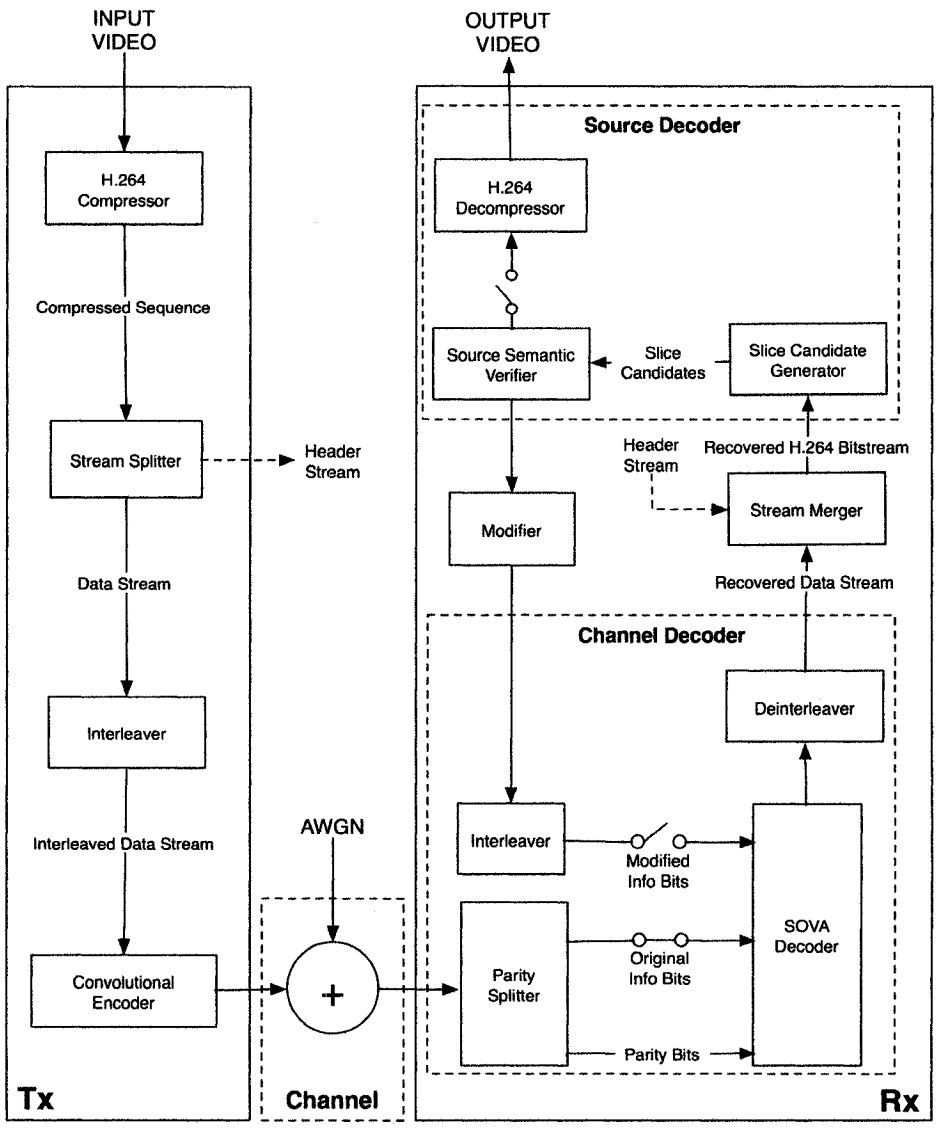

Figure 5.1: Block Diagram of the proposed scheme Iterative Joint Source-Channel Decoding (IJSCD)

When IJSCD is used to transmit a video sequence, the H.264 Compressor compresses the sequence and the Stream Splitter divides the sequence into a Header Stream and a Data Stream. Like ECSS, the proposed IJSCD scheme requires error-free transmission of all bits in layers above and including the slice header. The Data Stream is Interleaved and channel coded by the Convolutional Encoder. The channel-coded sequence is transmitted over an AWGN channel using BPSK modulation.

On the receiver side, the received sequence passes to the Channel Decoder where the Parity Splitter separates the information bits from the parity bits and passes both bitstreams to the SOVA decoder, which performs soft-output convolutional decoding. The Deinterleaver then places the decoded soft-bits in the original order of the bits in the Data Stream. The Stream Merger combines the error-free Header Stream with the recovered Data Stream and passes the result to the Source Decoder. In the Source Decoder, the SCG generates a set of slice candidates for each slice and the SSV chooses a winner among them. The winning slice candidate is passed to the Modifier, which alters the soft-values of the bits in the recovered Data Stream. The new soft-bit values are fed back into the Channel Decoder where an Interleaver places them in the same order as the original Information Bits and passes them to the SOVA decoder for the next iteration. On the last iteration, the Source Decoder decompresses the winning slice candidates to produce the output video sequence.

The Stream Splitter, the Stream Merger, the SCG and the SSV were discussed in Section 4.2 as part of ECSS and the H.264 Compressor and the H.264 Decompressor were discussed in Section 2.1.4 and Section 2.1.5 respectively. The Interleavers, the Deinterleaver, the Convolutional Encoder, the Parity Splitter, the SOVA Decoder and the Modifier are described in this section.

5.1.1 Interleaver and Deinterleaver

It was observed in Section 4.3.2 that it is difficult for the SSV to correct errors in a slice that contains a large number of bit errors. It is also known that burst errors often result when the SOVA Decoder selects the wrong decoding path. Because of this, errors in SOVA Decoding make it difficult for some slices to be corrected. However, using an Interleaver can alleviate this problem.

The Interleaver spreads the bits from each slice out among multiple information blocks. The Deinterleaver reconstructs the slices by redistributing the bits from each information block. Adjacent bits in an information block rarely come from the same slice. Thus, burst errors from the SOVA Decoder are spread out among different slices making it easier for the SSV to find a winning candidate for each slice.

Another advantage of interleaving is that improvements made by the SSV assist the SOVA Decoder and vice-versa. Since the SOVA Decoder operates on information blocks and the SSV operates on slices, when the SSV improves one slice, it affects multiple information blocks. Similarly, when the SOVA Decoder improves one information block, it affects multiple slices.

The Interleaver used on both the transmitter side and the receiver side is a block interleaver with a 5000-by-10 matrix. The Deinterleaver is a 10-by-5000 matrix that performs the inverse operation.

5.1.2 Convolutional Encoder

The channel code used in the proposed scheme is a rate-½ Recursive Systematic Convolutional (RSC) code with a constraint length of 2. The length of a convolutional code block is 20004 bits. Each code block consists of 10000 information bits, 10000 parity bits and 4 termination bits. Thus, each 50000-bit information block generated by the Interleaver produces 5 convolutional code blocks.

The generator matrix **G**(D) of the Convolutional Encoder is seen in Equation 5.1 and the controller canonical form is seen in Figure 5.2.

$$\mathbf{G}(D) = \left[1 \quad \frac{1+D^2}{1+D+D^2}\right] \tag{5.1}$$

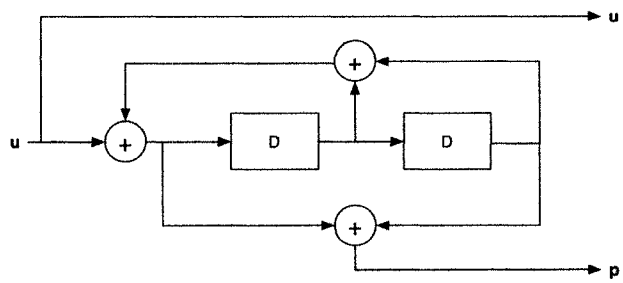

Figure 5.2: Controller canonical form of the Convolutional Encoder

Here, **u** represents the information bits, **p** represents the parity bits and D is a delay. The resulting output sequence, denoted **c**, consists of interspersing the information and parity bits as in Equation 5.2.

$$\mathbf{c} = [u_0, p_0, u_1, p_1, u_2, p_2, \ldots] \tag{5.2}$$

5.1.3 Parity Splitter

The Parity Splitter takes in a received channel code block and divides it into two streams: the Information Stream and the Parity Stream. Both streams contain soft valued bits. The Information Stream contains all information bits in the code block while the Parity Stream contains all parity bits. Both streams are passed to the SOVA Decoder. The SOVA Decoder uses the Parity Stream on all iterations of the IJSCD, but only uses the Information Stream on the first iteration of the IJSCD. On all subsequent iterations, a modified Information Stream from the Source Decoder is used.

5.1.4 SOVA Decoder

The Soft Output Viterbi Algorithm (SOVA) Decoder is a soft-input soft-output (SISO) Maximum-Likelihood (ML) convolutional decoder [40]. The soft-output bit values produced by the SOVA Decoder are needed by the SCG to generate slice candidates, as described in Section 4.2.2.

The SOVA Decoder assembles the channel-coded stream from the Information Stream and the Parity Stream. On the first IJSCD iteration, the Information Stream and the Parity Stream both come from the Parity Splitter, as mentioned in Section 5.1.3. On all subsequent iterations, the same Parity Stream is used and the Information Stream comes from the previous IJSCD iteration. The assembled stream is denoted **y**. Once assembled, the SOVA Decoder decodes the stream and produces soft-outputs, denoted $L(u_k)$, that correspond to the transmitted bits, denoted u_k.

To determine the value of soft-output $L(u_k)$, two likelihood estimates are defined. The first estimate, labeled p_k, is an estimate of the likelihood that a 1 was transmitted and is shown in Equation. The second estimate, labeled q_k, is an estimate of the likelihood that a 0 was transmitted. The values p_k and q_k are shown in Equations 5.3 and 5.4. They are similar to the likelihood values developed in Section 2.2, but are conditional on the entire received bitstream, not just a single received value.

$$p_k = P(u_k = 1 \mid \mathbf{y}) \tag{5.3}$$

$$q_k = 1 - p_k = P(u_k = 0 \mid \mathbf{y}) \tag{5.4}$$

$L(u_k)$ is defined as the logarithm likelihood ratio (LLR) of p_k and q_k, as shown in Equation 5.5.

$$L(u_k) = \log\left(\frac{p_k}{q_k}\right) \tag{5.5}$$

To determine the $L(u_k)$ values, the SOVA Decoder uses the classical Viterbi Algorithm [41] to find the ML codeword. As the ML codeword is decoded, a soft reliability measure is associated with each bit [40]. The reliability measure for each bit determines the magnitude of $L(u_k)$, and the corresponding bit value in the ML codeword determines the sign of $L(u_k)$.

5.1.5 Modifier

The Modifier takes in the winning slice candidate from the SSV, denoted **w**, and uses it to alter each of the $L(u_k)$ values originally decoded by the SOVA Decoder. If N is the length of the slice in bits, the winning slice candidate **w** is defined as:

$$\mathbf{w} = [w_0, w_1, w_2, ..., w_{N-1}], \quad \mathbf{w} \in [0,1]^N \tag{5.6}$$

Here, w_k is a bit in the winning slice candidate, and $k = 0,1,...N-1$. To alter $L(u_k)$, the likelihood estimates p_k and q_k are modified to include information about winning slice candidate **w**. Based on Equations 5.3 and 5.4, the resulting modified likelihood estimates, denoted $\hat{p}_k$ and $\hat{q}_k$ are given as:

$$\hat{p}_k = P(u_k = 1 \mid \mathbf{y}, \mathbf{w}) \tag{5.7}$$

$$\hat{q}_k = 1 - \hat{p}_k = P(u_k = 0 \mid \mathbf{y}, \mathbf{w}) \tag{5.8}$$

The altered value of $L(u_k)$, denoted $\hat{L}(u_k)$, is the LLR of $\hat{p}_k$ and $\hat{q}_k$. Referring to Equation 5.5, $\hat{L}(u_k)$ is:

$$\hat{L}(u_k) = \log\left(\frac{\hat{p}_k}{\hat{q}_k}\right) \tag{5.9}$$

It is difficult to determine an expression for the likelihood estimates in Equations 5.7 and 5.8 because it is difficult to quantify the contribution of winning slice candidate **w**. Thus, the values of $\hat{p}_k$ and $\hat{q}_k$ are determined in an ad-hoc manner in this thesis.

It is desired to have an expression for $\hat{p}_k$ that has contributions from both p_k and w_k. The contribution of w_k should be related to the certainty of the winning slice candidate **w**. The two extreme cases of **w** either being 0% certain or 100% certain are considered. If **w** is 0% certain, then w_k should contribute nothing and $\hat{p}_k$ should be equal to p_k. On the other hand, if **w** is 100% certain, then p_k should contribute nothing and $\hat{p}_k$ should be equal to w_k ($\hat{p}_k$ should be 0 when w_k is 0 and 1 when w_k is 1). If **w** is neither 0% nor 100% certain, but somewhere in between, both p_k and w_k should have an effect on the value of $\hat{p}_k$. Given these guidelines, the following expression was considered:

$$\hat{p}_k = (1-\alpha)\cdot p_k + \alpha \cdot w_k \tag{5.10}$$

Here, α is called the Modification Parameter and shall be determined empirically. The value of α for slice candidates that pass semantic verification will be different from the value of α for slice candidates that fail semantic verification. The value of α is related to the certainty of the bits in slice candidate **w** and can be any real number between 0 and 1 inclusively.

Clearly, $\hat{p}_k$ is unaffected by w_k when α is 0 and unaffected by p_k when α is 1. When α is between 0 and 1, both p_k and w_k affect the value of $\hat{p}_k$. As α increases, the influence of p_k diminishes and the influence of w_k increases. Also, $\hat{p}_k$ is less than p_k when w_k is 0 and greater than p_k when w_k is 1 (i.e. p_k is modified in the direction of w_k). Combining Equations 5.8 and 5.10, the expression for $\hat{q}_k$ is developed as:

$$\hat{q}_k = (1-\alpha)\cdot q_k + \alpha \cdot (1-w_k) \tag{5.11}$$

An expression for $\hat{L}(u_k)$ can now be presented. Combining Equations 5.9, 5.10 and 5.11, $\hat{L}(u_k)$ is written as:

$$\begin{aligned}\hat{L}(u_k) &= \log\left(\frac{(1-\alpha)\cdot p_k + \alpha \cdot w_k}{(1-\alpha)\cdot q_k + \alpha \cdot (1-w_k)}\right) \\ &= \log\left(\frac{p_k + \alpha \cdot (w_k - p_k)}{q_k - \alpha \cdot (w_k - p_k)}\right)\end{aligned} \tag{5.12}$$

To simplify Equation 5.12, Equations 5.3 and 5.4 are used to separate the expression for $\hat{L}(u_k)$ into one part for $w_k = 0$ and one part for $w_k = 1$.

$$
\begin{aligned}
L(\hat{u}_k) &= \begin{cases} \log\left(\dfrac{p_k - \alpha \cdot p_k}{q_k + \alpha \cdot p_k}\right), & w_k = 0 \\ \log\left(\dfrac{p_k + \alpha \cdot q_k}{q_k - \alpha \cdot q_k}\right), & w_k = 1 \end{cases} \\
&= \begin{cases} \log\left(\dfrac{1-\alpha}{q_k/p_k + \alpha}\right), & w_k = 0 \\ \log\left(\dfrac{p_k/q_k + \alpha}{1-\alpha}\right), & w_k = 1 \end{cases} \\
&= \begin{cases} -\log\left(\dfrac{(p_k/q_k)^{-1} + \alpha}{1-\alpha}\right), & w_k = 0 \\ \log\left(\dfrac{p_k/q_k + \alpha}{1-\alpha}\right), & w_k = 1 \end{cases}
\end{aligned}
\tag{5.13}
$$

The two parts in Equation 5.13 can now be recombined as:

$$
\hat{L}(u_k) = (2w_k - 1) \cdot \log\left(\frac{(p_k/q_k)^{(2w_k-1)} + \alpha}{1-\alpha}\right) \tag{5.14}
$$

By manipulating Equation 5.5, the ratio p_k/q_k is determined as:

$$
p_k/q_k = e^{L(u_k)} \tag{5.15}
$$

Substituting Equation 5.15 into Equation 5.14, $\hat{L}(u_k)$ becomes:

$$
\hat{L}(u_k) = (2w_k - 1) \cdot \log\left(\frac{e^{(2w_k-1)\cdot L(u_k)} + \alpha}{1-\alpha}\right) \tag{5.16}
$$

The Modifier uses Equation 5.16 to alter the value of $L(u_k)$ and produce $\hat{L}(u_k)$. $L(u_k)$ is pushed towards 0 (and often passes through it) when bit k is a flip-bit. On the other hand, $L(u_k)$ is pushed away from 0 (thus reinforcing its hard decision) when bit k is a non flip-bit.

In the end, the Modified soft $\hat{L}(u_k)$ values are fed back into the Channel Decoder, where they are interleaved and passed to the SOVA Decoder as the Information Stream for the next IJSCD iteration.

5.2 Performance Evaluation

The performance of the proposed IJSCD scheme is compared to the performance of convolutional decoding by itself. Like the performance evaluation for ECSS in Section 4.3, the performance here is measured both objectively using PSNR and BER, and subjectively by looking at several decompressed video sequences. For the reader's own assessment, the subjective performance evaluation includes a few sample frames that are decompressed by both IJSCD scheme and the convolutional decoding scheme. The complexity of the IJSCD scheme is also measured and discussed.

In all of the experiments performed, the number of slice candidates generated by the SCG is 300 and the same compression parameters as in Section 4.3 are used. The only parameter that needs to be determined is the Modification Parameter α, which is determined empirically prior to the objective and subjective performance evaluations and the complexity analysis.

5.2.1 Determination of Modification Parameter

The choice of the Modification Parameter α does not affect the complexity of the proposed scheme, but does affect performance. Thus, the effect of the Modification Parameter on the performance of the proposed scheme is investigated. As discussed in Section 5.1.5, there are two cases to investigate: when the winning slice candidate had passed semantic verification and when the winning slice candidate had failed semantic verification. The Modification Parameter is denoted α_1 for the first case and α_2 for the second case.

While α_1 is applied to all bits in the slice, α_2 is only applied to flip-bits. When a winning slice candidate fails semantic verification, one or more of its non-flip-bits must be erroneous. Modifying the non-flip-bits will make the erroneous bits worse. On the other hand, the results of Section 4.3.1 show that flipping the flip-bits improves performance even when the winning candidate fails semantic verification. Thus, only the flip-bits will be modified in winning slice candidates that fail semantic verification.

An experiment was performed to select the values of α_1 and α_2. Both α_1 and α_2 should be between 0 and 1. From past simulations, it was found that the values of α_1 and α_2 should be close to 1. In the experiment, the value of α_1 is determined first. To do so,

the proposed scheme is simulated using different α_1 values while α_2 is fixed at 0.8. Once α_1 is set, the value of α_2 is determined. The proposed scheme is simulated using different α_2 values while α_1 is fixed at 0.95. The PSNR of the output videos in both cases is observed.

The experiments are run on the video sequences "Football", "Table-Tennis" and "Mobile" at a channel SNR of 1.8dB. In the experiment, 2 iterations of the proposed scheme are performed and the SCG generates 300 slice candidates. The experiments are run 10 times at each data point with independent seeds.

5.2.1.1 Determination of α_1

The first experiment was run on the proposed scheme as described with $\alpha_2 = 0.8$. The PSNR (after decompression) and BER (prior to decompression) were measured at several values of α_1 for each sequence tested. For brevity, only the luminance PSNR values are shown. The red and blue chrominance PSNR and BER values show similar results. Figure 5.3, Figure 5.4 and Figure 5.5 show the luminance PSNR for the video sequences "Football", "Table-Tennis" and "Mobile" respectively.

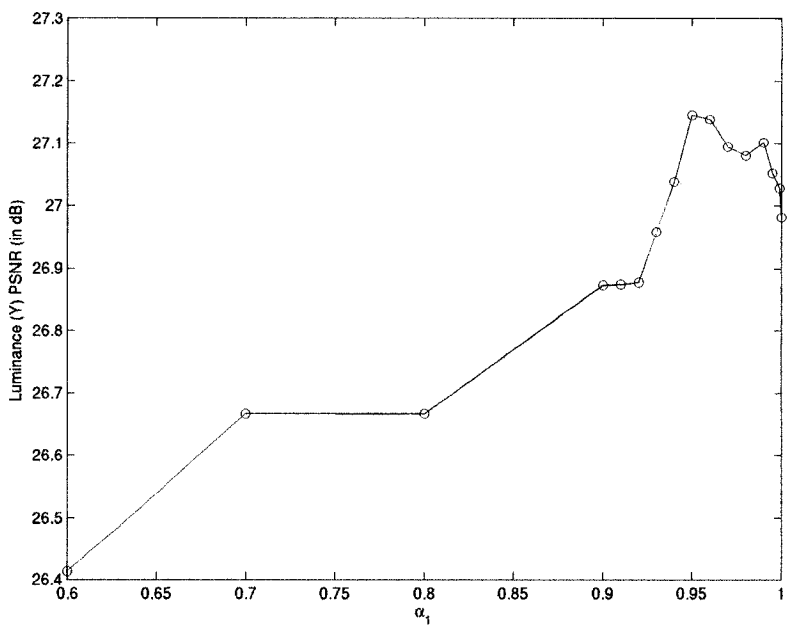

Figure 5.3: Luminance PSNR of video "Football" vs. Modification Parameter α_1 with $\alpha_2 = 0.8$

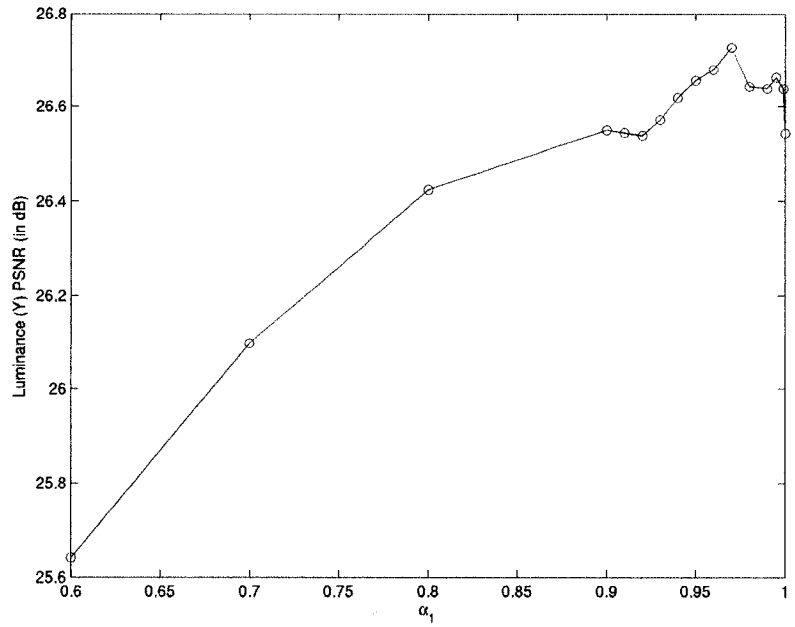

Figure 5.4: Luminance PSNR of video "Table-Tennis" vs. Modification Parameter α_1 with $\alpha_2 = 0.8$

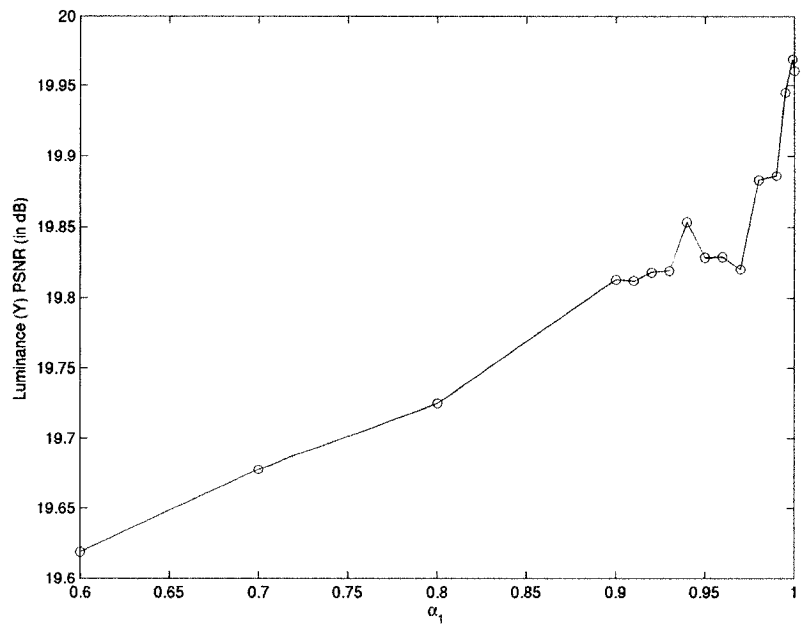

Figure 5.5: Luminance PSNR of video "Mobile" vs. Modification Parameter α_1 with $\alpha_2 = 0.8$

While there are differences between the 3 curves presented, there are some similarities. It is noted that the luminance PSNR increases significantly as a function of α_1 until approximately $\alpha_1 = 0.9$. Between $\alpha_1 = 0.9$ and $\alpha_1 = 0.999$, the PSNR oscillates

differently in each of the 3 curves. However, the amplitude of the oscillations is relatively small. This indicates that the value of α_1 should not have a serious negative impact on the results if it is chosen to be somewhere between 0.9 and 0.999.

The value of α_1 is chosen to be 0.95, which yield good results on average for all video sequences tested. The choice of α_1 is based on empirical observations and may not necessarily be the best choice. In the future, a better method of selecting α_1 may further improve performance.

5.2.1.2 Determination of α_2

The second experiment was run on the proposed scheme as described with $\alpha_1 = 0.95$. Similar to the previous experiment to select α_1, the PSNR and BER were measured at several values of α_2 for each sequence tested. Again for brevity, only the luminance PSNR values are shown as the chrominance PSNR and BER values show similar results. Figure 5.6, Figure 5.7 and Figure 5.8 show the luminance PSNR for the video sequences "Football", "Table-Tennis" and "Mobile" respectively.

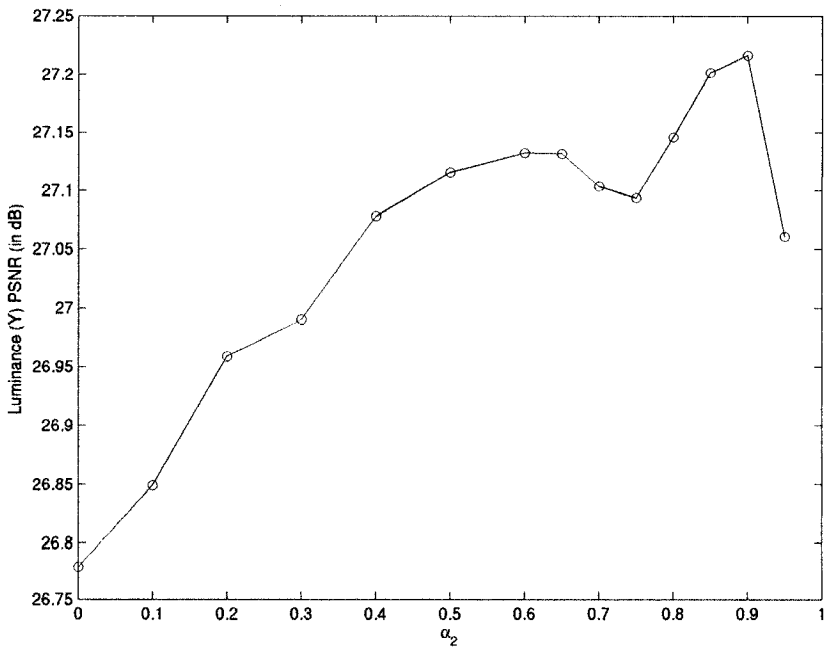

Figure 5.6: Luminance PSNR of video "Football" vs. Modification Parameter α_2 with $\alpha_1 = 0.95$

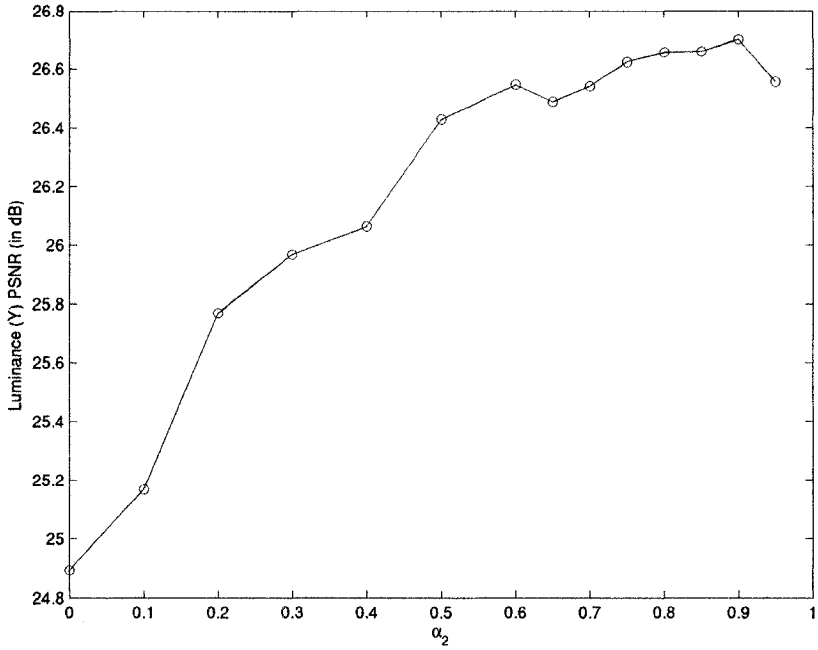

Figure 5.7: Luminance PSNR of video "Table-Tennis" vs. Modification Parameter α_2 with $\alpha_1 = 0.95$

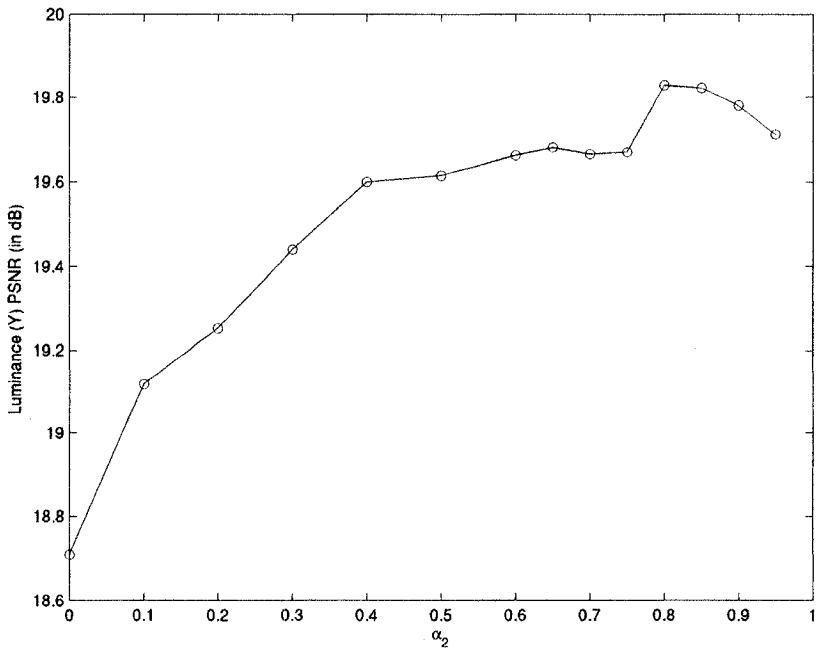

Figure 5.8: Luminance PSNR of video "Mobile" vs. Modification Parameter α_2 with $\alpha_1 = 0.95$

It is observed from the 3 curves that the luminance PSNR peaks between $\alpha_2 = 0.8$ and $\alpha_2 = 0.9$ depending on the video sequence. As was the case for α_1, the PSNR

oscillates differently in each of the 3 curves, but the amplitude of the oscillations is relatively small. As such, the choice of α_2 should not have a serious impact on the results if chosen to be between 0.8 and 0.9.

The value of α_2 is chosen to be 0.8, which yields good results on average for all video sequences tested. As with the choice of α_1, the choice of α_2 is based on empirical observations and may not necessarily be the best choice. It is possible that a different value of α_2 may improve performance.

5.2.2 Objective Performance

With $\alpha_1 = 0.95$ and $\alpha_2 = 0.8$ as determined in Section 5.2.1, the performance of the proposed IJSCD scheme is evaluated using PSNR and BER as in Section 4.3.2. Three sets of simulations are done to evaluate the performance of the scheme. The first set of simulations is done using the video sequence "Football", the second set is done using the video "Table-Tennis" and the third set is done using the video "Foreman". In each set of simulations, the IJSCD scheme uses 300 slice candidates and the performance of IJSCD is compared to the performance of the convolutional code by itself.

5.2.2.1 Objective Performance for Video "Football"

The 4-second video sequence "Football" is encoded using the same compression parameters as in Section 4.3.2.1 and has the same maximum achievable PSNR. Simulations are run 10 times with independent seeds at several channel SNR values. The result for each channel SNR value is the average of the 10 simulation runs.

Figure 5.9 shows the objective performance comparison in terms of luminance PSNR and Figure 5.10 shows the BER of the output compressed video sequence before it is decompressed. Results are shown for IJSCD with 1, 2, 3 and 4 iterations, and are compared to the results of the convolutional coding scheme. As in Section 4.3.2.1, the results for red and blue chrominance PSNR are similar to the results for luminance PSNR and, for brevity, are not shown.

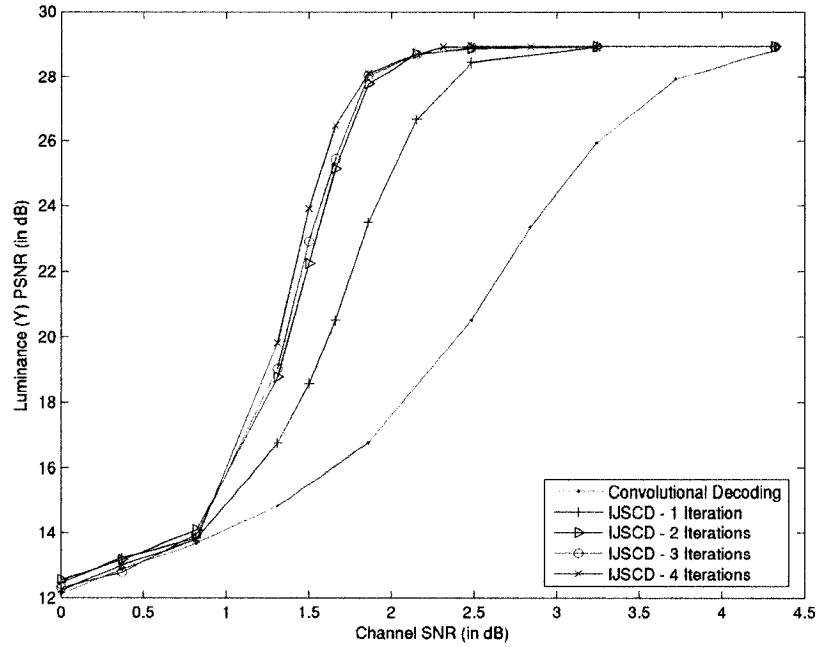

Figure 5.9: Luminance (Y) PSNR vs. channel SNR for video "Football"

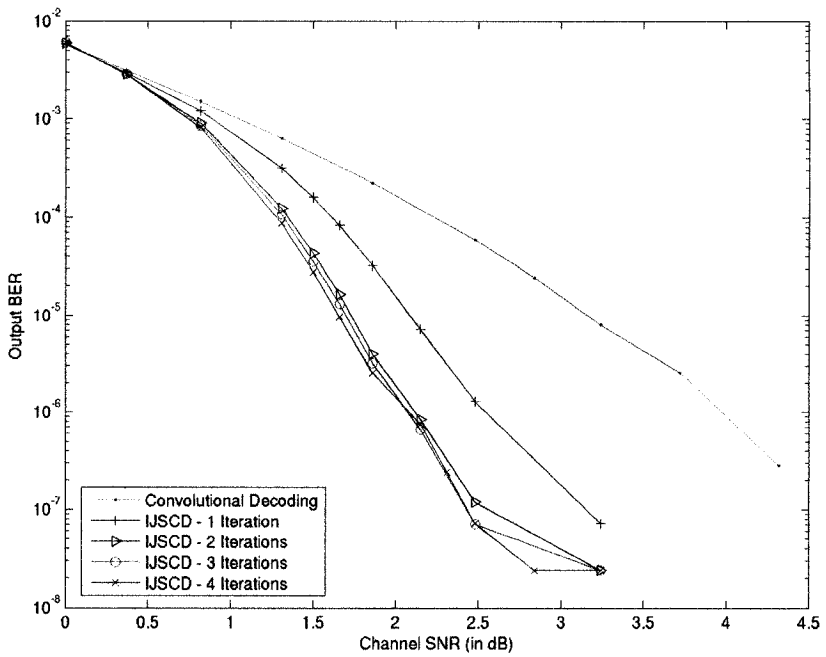

Figure 5.10: Output Bit Error Rate (BER) vs. channel SNR for video "Football"

As in Section 4.3.2, the minimum channel SNR needed to achieve the maximum PSNR is used to measure the performance gain of the IJSCD scheme. Table 5.1 compares

the minimum channel SNR needed with convolutional decoding and the minimum channel SNR needed with IJSCD using 1, 2, 3 and 4 iterations.

Scheme	Channel SNR
Convolutional Decoding	4.4 dB
IJSCD – 1 Iteration	3.0 dB
IJSCD – 2 Iterations	2.4 dB
IJSCD – 3 Iterations	2.3 dB
IJSCD – 4 Iterations	2.3 dB

Table 5.1: Minimum Channel SNR needed to achieve maximum luminance (Y) PSNR for video sequence "Football"

With convolutional decoding, the maximum achievable PSNR is obtained with a channel SNR of 4.4dB or higher. In comparison, IJSCD can obtain the maximum achievable PSNR with a channel SNR of 3.0dB, 2.4 dB and 2.3dB with 1, 2 and 3 iterations respectively. Thus, IJSCD provides a gain in channel SNR of 1.4dB, 2.0dB and 2.1dB with 1, 2 and 3 iterations respectively, as seen in Table 5.2.

# Iterations	Gain in Channel SNR
1	1.4 dB
2	2.0 dB
3	2.1 dB
4	2.1 dB

Table 5.2: Performance gain of IJSCD vs. convolutional decoding at maximum achievable PSNR for video sequence "Football"

When the IJSCD scheme uses 4 iterations, the maximum PSNR is achieved at the same minimum channel SNR as with 3 iterations. However, Figure 5.9 shows that when the channel SNR is between 1dB and 2.2dB, IJSCD yields a slightly higher PSNR with 4 iterations than with 3 iterations.

5.2.2.2 Objective Performance for Video "Table-Tennis"

The 4-second video sequence "Table-Tennis" is encoded using the same compression parameters as in Section 4.3.2.2 and has the same maximum achievable PSNR. Like

"Football", simulations are run 10 times with independent seeds at several channel SNR values and the average of the 10 simulation runs is taken.

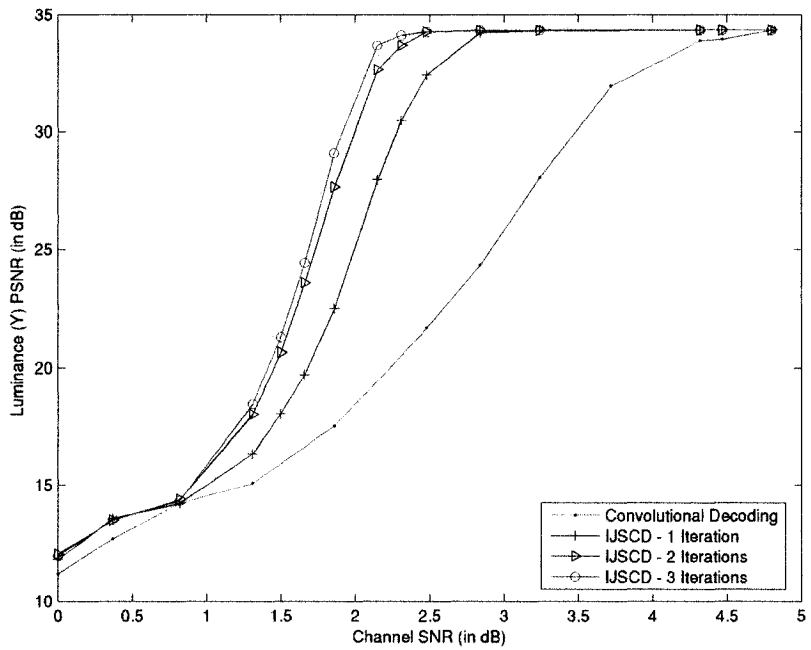

Figure 5.11: Luminance (Y) PSNR vs. channel SNR for video "Table-Tennis"

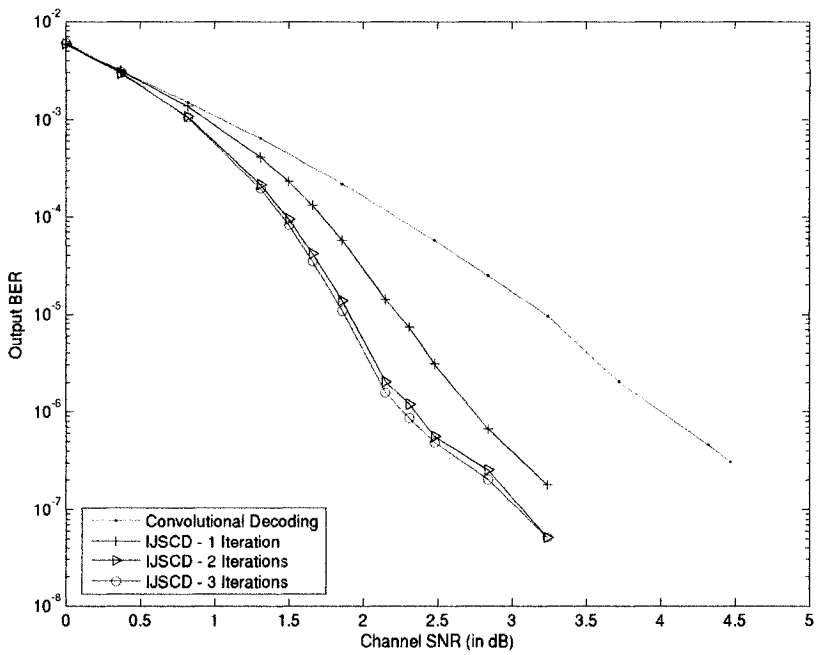

Figure 5.12: Output Bit Error Rate (BER) vs. channel SNR for video "Table-Tennis"

Figure 5.11 shows the performance comparison for luminance PSNR and Figure 5.12 shows the performance comparison for BER. Results are shown for IJSCD using 1, 2 and 3 iterations and, as with "Football", are compared to the results for convolutional decoding. Like in the previous section, the chrominance PSNR curves are similar to the luminance PSNR curve and are not shown.

Scheme	Channel SNR
Convolutional Decoding	4.4 dB
IJSCD – 1 Iteration	2.9 dB
IJSCD – 2 Iterations	2.5 dB
IJSCD – 3 Iterations	2.35 dB

Table 5.3: Minimum Channel SNR needed to achieve maximum luminance (Y) PSNR for video sequence "Table-Tennis"

Table 5.3 shows the minimum channel SNR needed to achieve the maximum luminance PSNR for convolutional decoding and for IJSCD. Table 5.4 shows the resulting gains in channel SNR at the maximum achievable PSNR provided by the proposed (for each number of slice candidates tested) over convolutional decoding. The gains in channel SNR for IJSCD are similar to the gains seen for "Football".

# Iterations	Gain in Channel SNR
1	1.5 dB
2	1.9 dB
3	2.05 dB

Table 5.4: Performance gain of IJSCD vs. convolutional decoding at maximum achievable PSNR for video sequence "Table-Tennis"

5.2.2.3 Objective Performance for Video "Foreman"

The 4-second video sequence "Foreman" differs from "Football" and "Table-Tennis" because it has a frame size of 352x288. Other than the frame size, the same compression parameters as "Football" and "Table-Tennis" are used (including a 1Mb/s bitrate).

Figure 5.13 shows the performance comparison for luminance PSNR and Figure 5.14 shows the performance comparison for BER. Results are shown for IJSCD using 1, 2 and

3 iterations and for convolutional decoding. Again, the chrominance PSNR curves are similar to the luminance PSNR curve and are not shown.

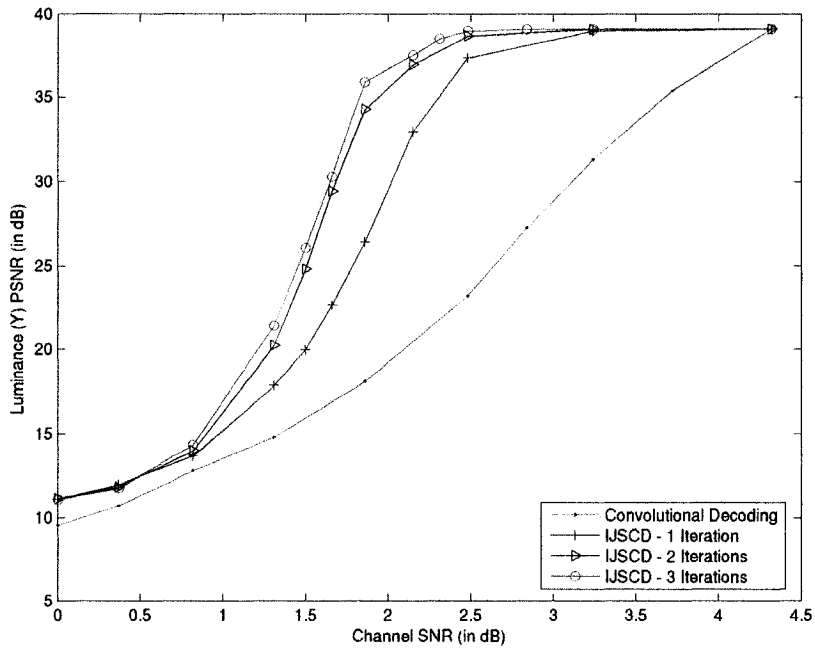

Figure 5.13: Luminance (Y) PSNR vs. channel SNR for video "Foreman"

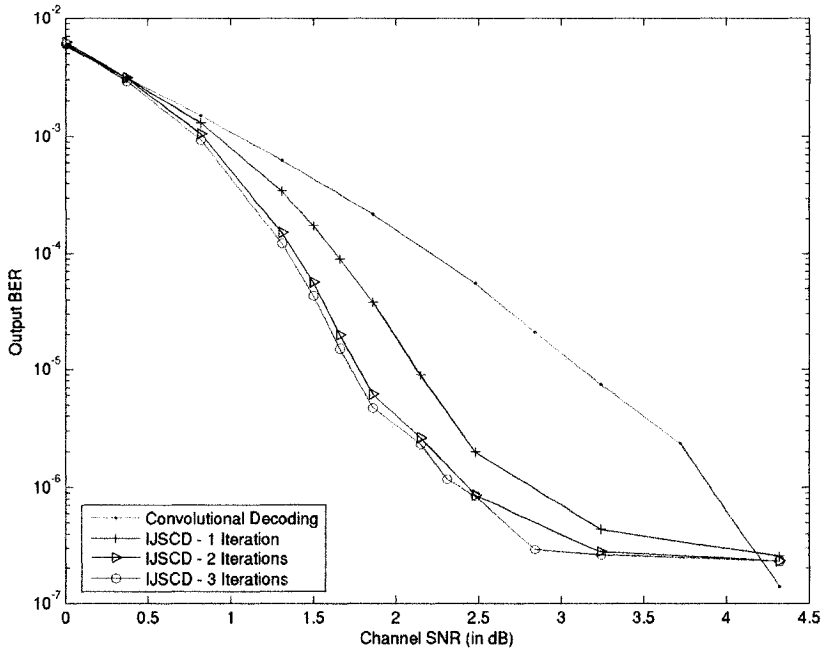

Figure 5.14: Output Bit Error Rate (BER) vs. channel SNR for video "Foreman"

Table 5.5 shows the minimum channel SNR needed to achieve the maximum luminance PSNR for convolutional decoding and for IJSCD. Table 5.6 shows the resulting gains in channel SNR provided by IJSCD (for each number of slice candidates tested) over convolutional decoding.

Scheme	Channel SNR
Convolutional Decoding	4.4 dB
IJSCD – 1 Iteration	3.2 dB
IJSCD – 2 Iterations	2.7 dB
IJSCD – 3 Iterations	2.45 dB

Table 5.5: Minimum Channel SNR needed to achieve maximum luminance (Y) PSNR for video sequence "Foreman"

# Iterations	Gain in Channel SNR
1	1.2 dB
2	1.7 dB
3	1.95 dB

Table 5.6: Performance gain of IJSCD vs. convolutional decoding for video sequence "Foreman"

The gains in channel SNR for IJSCD using 1, 2 and 3 iterations are slightly lower for "Foreman" than they are for either "Football" or "Table-Tennis". This could be due to the fact that the frame size is different while the bitrate is the same. In any case, the gains are still significant.

5.2.3 Subjective Performance

The subjective performance of the proposed IJSCD scheme is evaluated by looking at decompressed video sequences. The video sequences "Football" and "Table-Tennis" are used in the subjective performance evaluation. The same compression parameters that were used in the objective performance evaluation are used here.

To evaluate the performance of the IJSCD scheme subjectively, its resulting decompressed video was compared to the resulting decompressed video from convolutional decoding at the same channel SNR. IJSCD was observed using several different numbers of iterations. This section presents comments based on subjective observations as well as several decompressed frames for the reader's own assessment.

5.2.3.1 Subjective Performance for Video "Football"

When looking at the decompressed video "Football", there are obvious improvements in picture quality when IJSCD is used in comparison to convolutional decoding. There are also improvements in picture quality when the number of iterations increases from 1 to 2 and from 2 to 3. The differences between 2 and 3 iterations are, however, very small. There are rarely any noticeable improvements in picture quality between 3 and 4 iterations.

At 2.3dB channel SNR, the decompressed video using convolutional decoding is blocky and a significant portion of each frame is lost. In contrast, the decompressed video using 1 IJSCD iteration at 2.3dB channel SNR is mostly viewable, although several frames have visible errors. When 2 iterations are performed, hardly any frames have visible errors and each error is less severe on average than when 1 iteration is performed. There is hardly any noticeable improvement in video quality between 2 and 3 iterations at 2.3dB channel SNR and most of the frames in both cases are completely error free.

For the reader's own assessment, frames 42 and 115 are presented after having been transmitted over an AWGN at 2.3dB and after having been decompressed using convolutional decoding and IJSCD with 1 and 2 iterations. For comparative purposes, the error-free frames are also presented. Figure 5.15 to Figure 5.18 show the results for frame 42 using each of the schemes. Figure 5.19 to Figure 5.22 show the results for frame 115 using each of the schemes. The luminance PSNR values of frames 42 and 115 that correspond to each of the decoding schemes and the luminance PSNR of the error-free case are presented in Table 5.7 and Table 5.8 respectively. The decompressed frames and the PSNR values both show that performance is improved by IJSCD with 1 iteration and further improved with 2 iterations.

Scheme	PSNR (Y)
Error-free frame	28.53 dB
Convolutional Decoding	16.17 dB
IJSCD – 1 Iteration	25.22 dB
IJSCD – 2 Iterations	28.53 dB

Table 5.7: Luminance PSNR of decompressed frame 42 of video sequence "Football" transmitted over AWGN channel at 2.3dB channel SNR

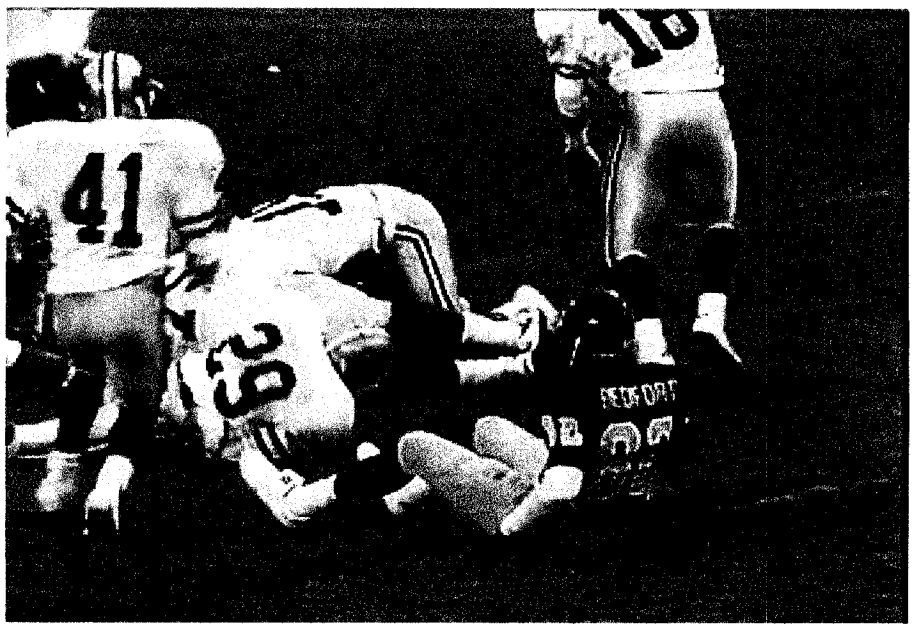

Figure 5.15: Decompressed error-free frame 42 of video sequence "Football"

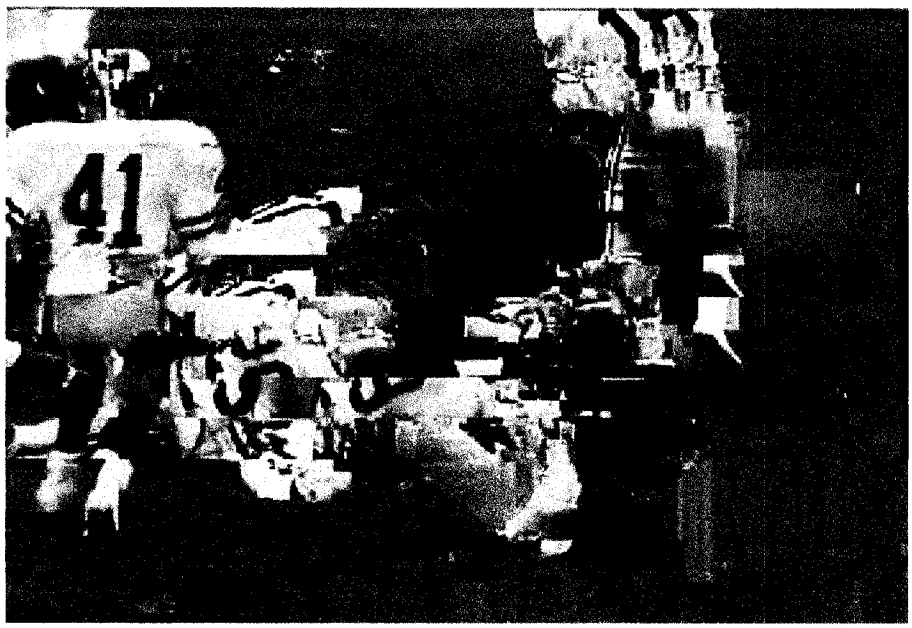

Figure 5.16: Frame 42 of video sequence "Football" transmitted over AWGN channel at 2.3dB SNR and decoded using convolutional decoding

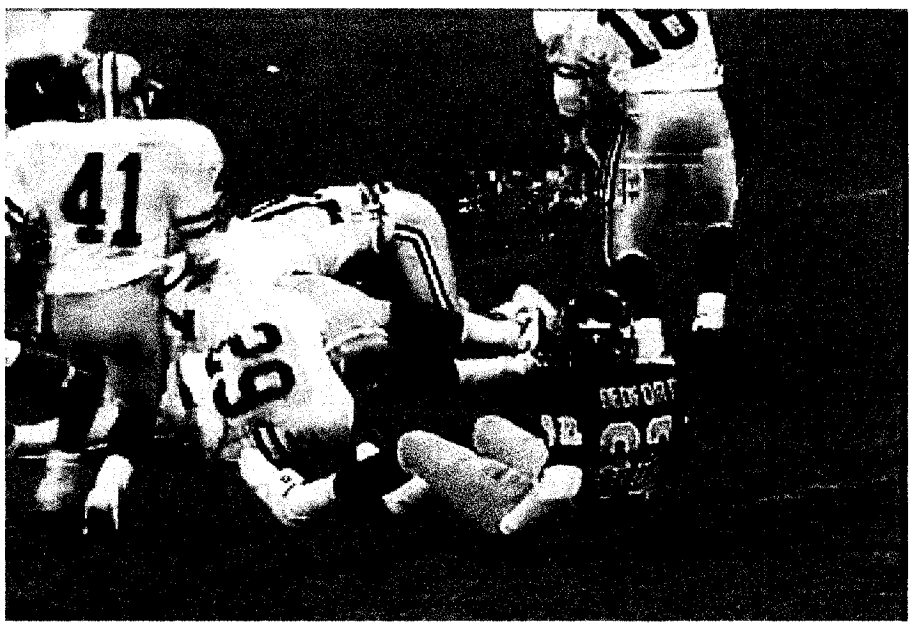

Figure 5.17: Frame 42 of video sequence "Football" transmitted over AWGN channel at 2.3dB SNR and decoded using IJSCD with 1 iteration

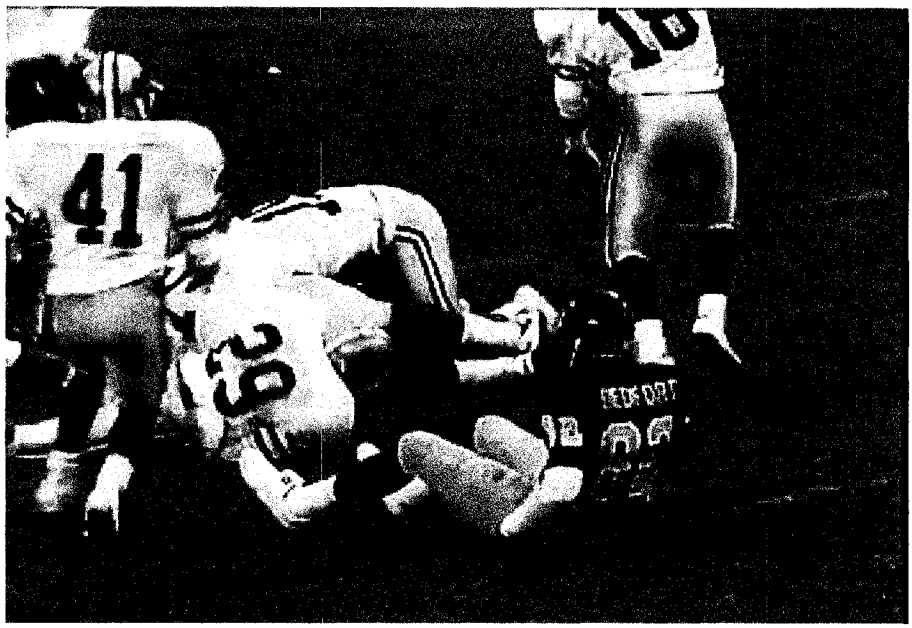

Figure 5.18: Frame 42 of video sequence "Football" transmitted over AWGN channel at 2.3dB SNR and decoded using IJSCD with 2 iterations

Scheme	PSNR (Y)
Error-free frame	29.11 dB
Convolutional Decoding	19.06 dB
IJSCD – 1 Iteration	22.97 dB
IJSCD – 2 Iterations	29.11 dB

Table 5.8: Luminance PSNR of decompressed frame 115 of video sequence "Football" transmitted over AWGN channel at 2.3dB channel SNR

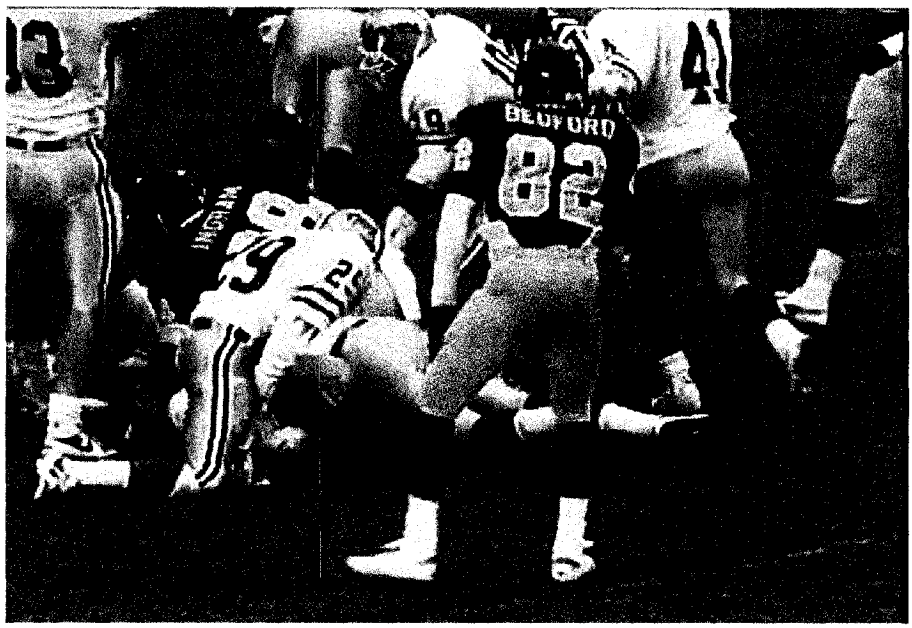

Figure 5.19: Decompressed error-free frame 115 of video sequence "Football"

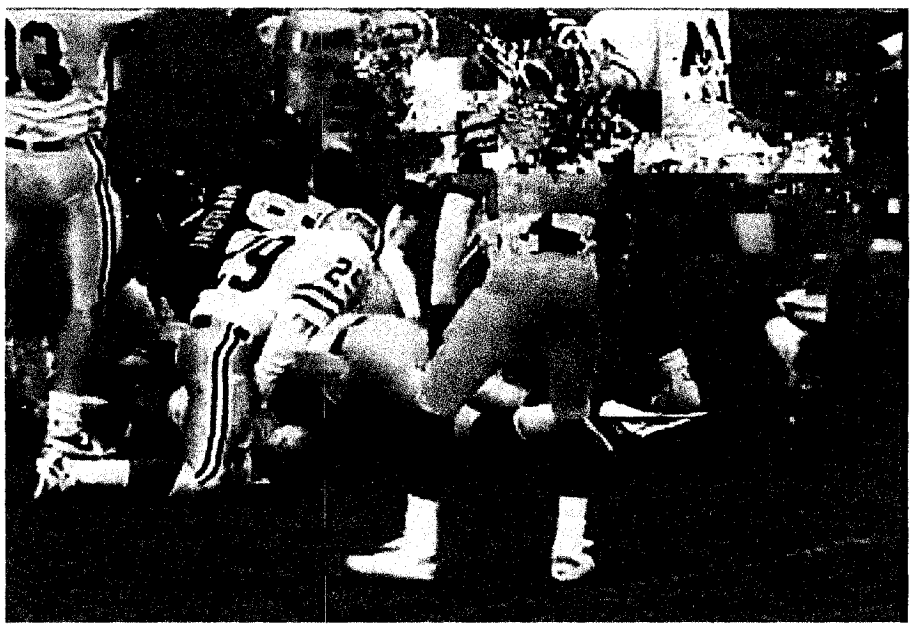

Figure 5.20: Frame 115 of video sequence "Football" transmitted over AWGN channel at 2.3dB SNR and decoded using convolutional decoding

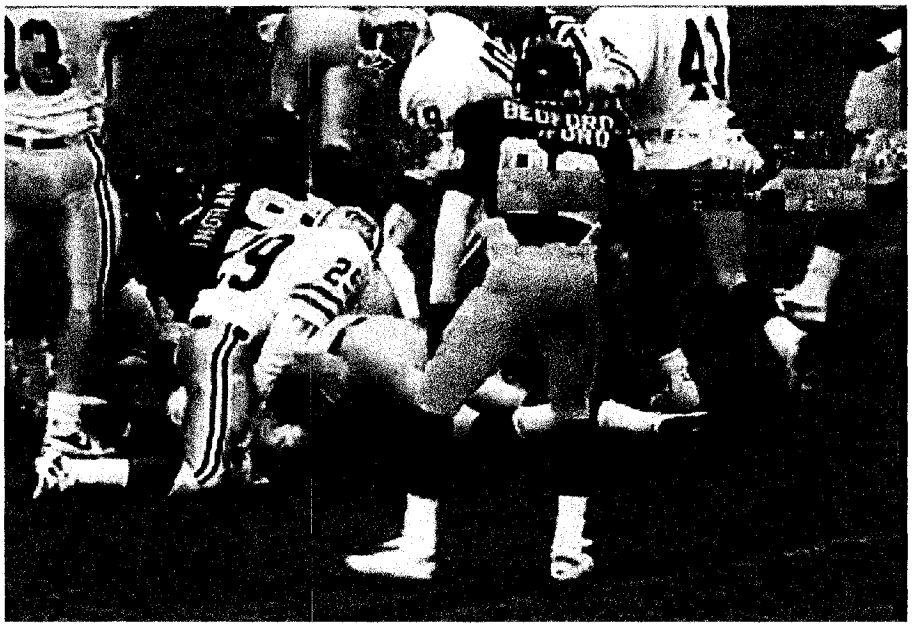

Figure 5.21: Frame 115 of video sequence "Football" transmitted over AWGN channel at 2.3dB SNR and decoded using IJSCD with 1 iteration

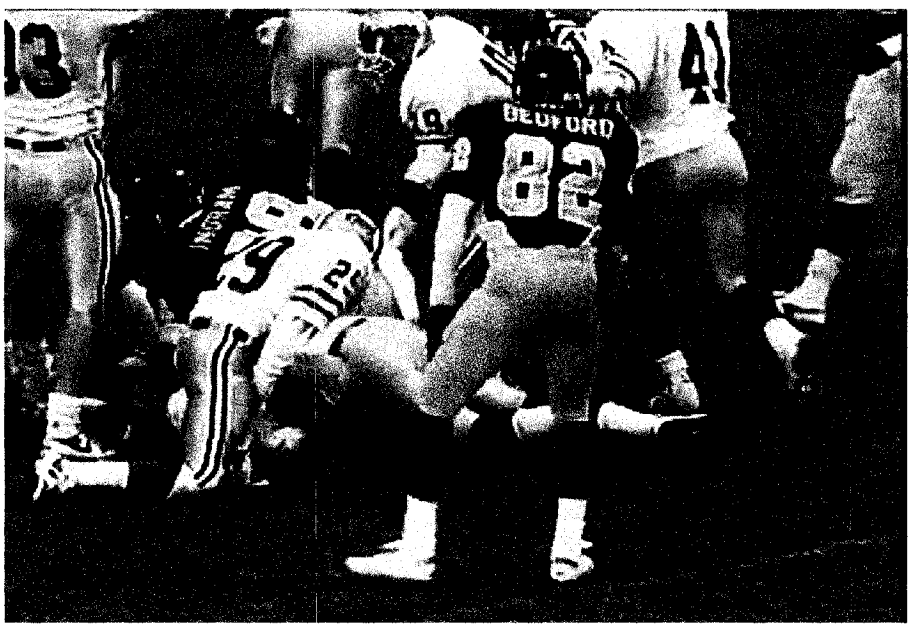

Figure 5.22: Frame 115 of video sequence "Football" transmitted over AWGN channel at 2.3dB SNR and decoded using IJSCD with 2 iterations

5.2.3.2 Subjective Performance for Video "Table-Tennis"

Like "Football", the video sequence "Table-Tennis" is evaluated at 2.3dB channel SNR. Once again, there are clear improvements in picture quality when IJSCD is used in

comparison to convolutional decoding. Also, there are improvements in picture quality when the number of iterations is increased from 1 to 2. With two iterations, most of the frames have no noticeable errors in them and there are rarely any noticeable improvements between 2 and 3 iterations.

For the reader's own assessment, frames 40 and 83 are presented after having been transmitted over an AWGN at 2.3dB and after having been decompressed using convolutional decoding as well as IJSCD. The error-free frames are also presented. Figure 5.23 to Figure 5.26 show the results for frame 40 using each of the schemes and Figure 5.27 to Figure 5.30 show the results for frame 83 using each of the schemes. The corresponding component PSNR values for each of the decoding schemes are presented in Table 5.9 and Table 5.10 respectively.

As was observed for "Football", the output frames and the PSNR values both show that performance is improved by IJSCD with 1 iteration and further improved with two iterations.

Scheme	PSNR (Y)
Error-free frame	36.41 dB
Convolutional Decoding	22.40 dB
IJSCD – 1 Iteration	28.66 dB
IJSCD – 2 Iterations	36.41 dB

Table 5.9: Luminance PSNR of decompressed frame 40 of video sequence "Table-Tennis" transmitted over AWGN channel at 2.3dB channel SNR

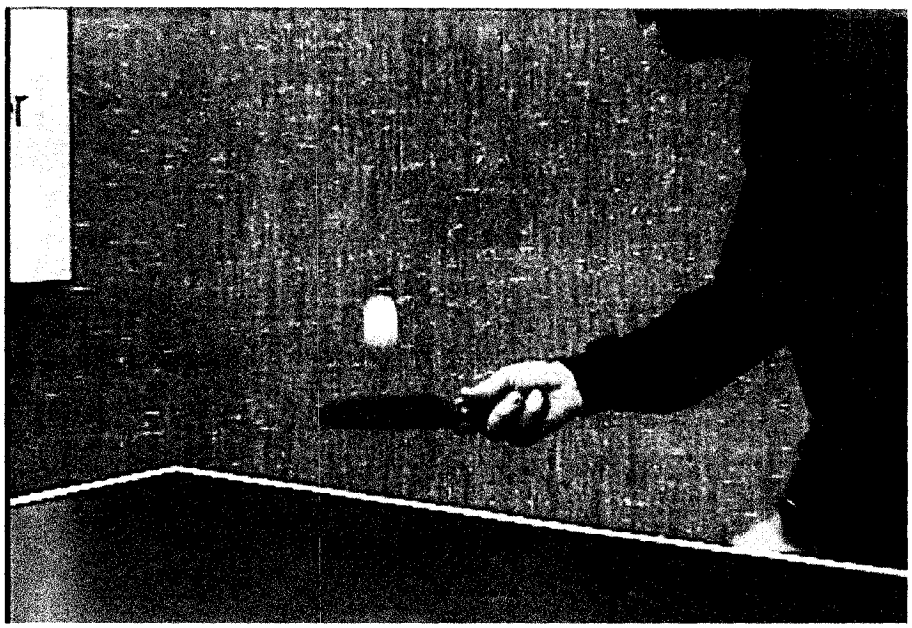

Figure 5.23: Decompressed error-free frame 40 of video sequence "Table-Tennis"

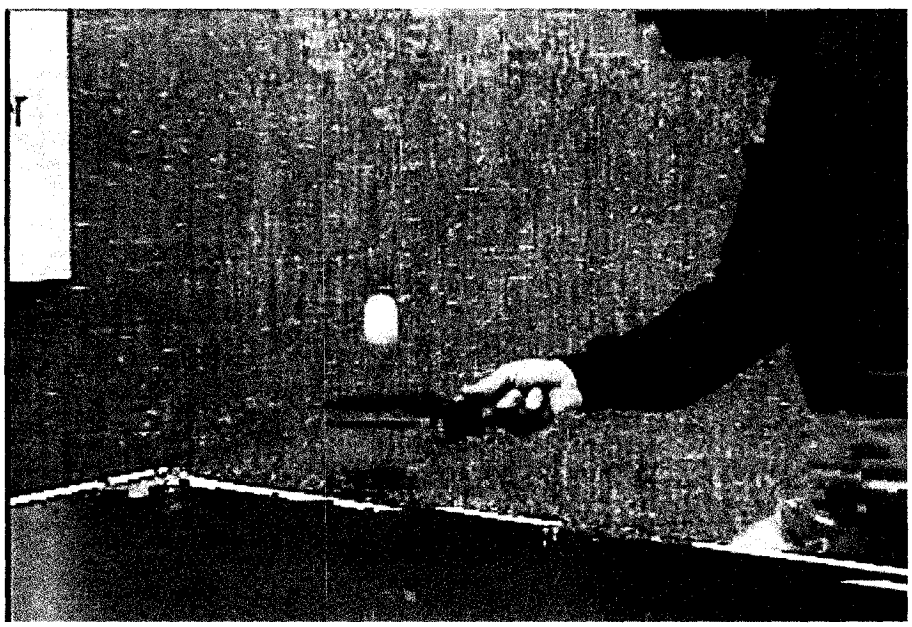

Figure 5.24: Frame 40 of video sequence "Table-Tennis" transmitted over AWGN channel at 2.3dB SNR and decoded using convolutional decoding

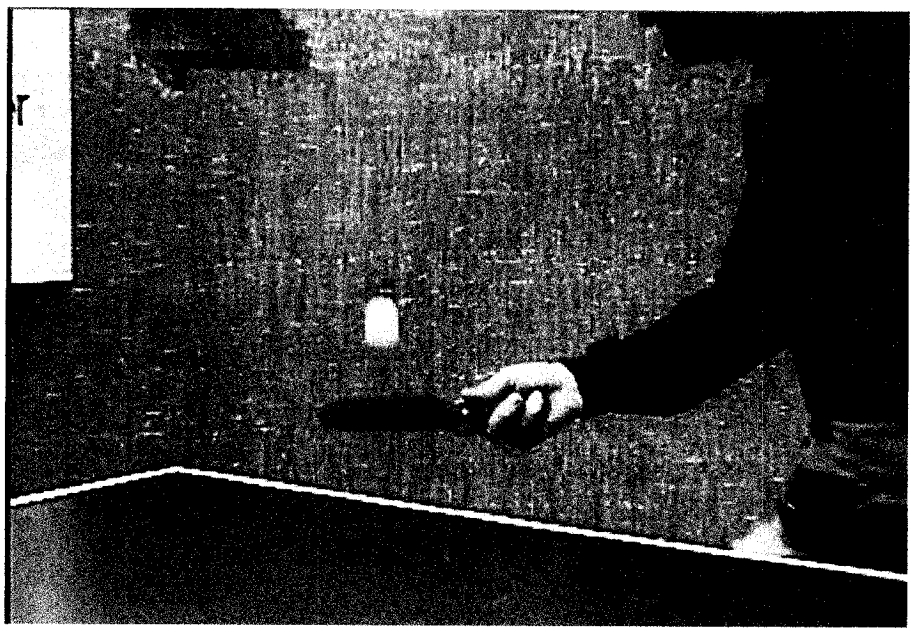

Figure 5.25: Frame 40 of video sequence "Table-Tennis" transmitted over AWGN channel at 2.3dB SNR and decoded using IJSCD with 1 iteration

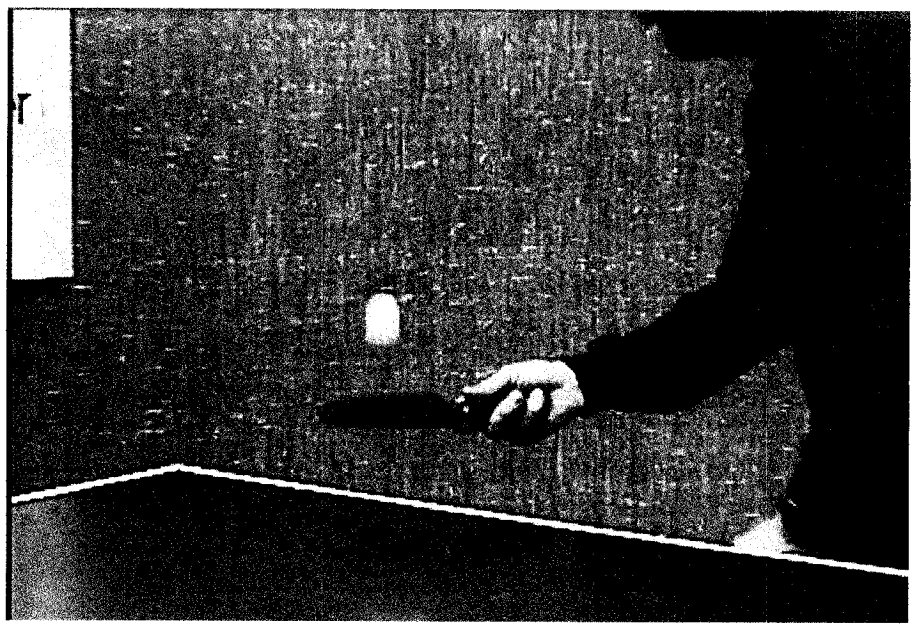

Figure 5.26: Frame 40 of video sequence "Table-Tennis" transmitted over AWGN channel at 2.3dB SNR and decoded using IJSCD with 2 iterations

Scheme	PSNR (Y)
Error-free frame	38.35 dB
Convolutional Decoding	19.51 dB
IJSCD – 1 Iteration	32.68 dB
IJSCD – 2 Iterations	38.35 dB

Table 5.10: Luminance PSNR of decompressed frame 83 of video sequence "Table-Tennis" transmitted over AWGN channel at 2.3dB channel SNR

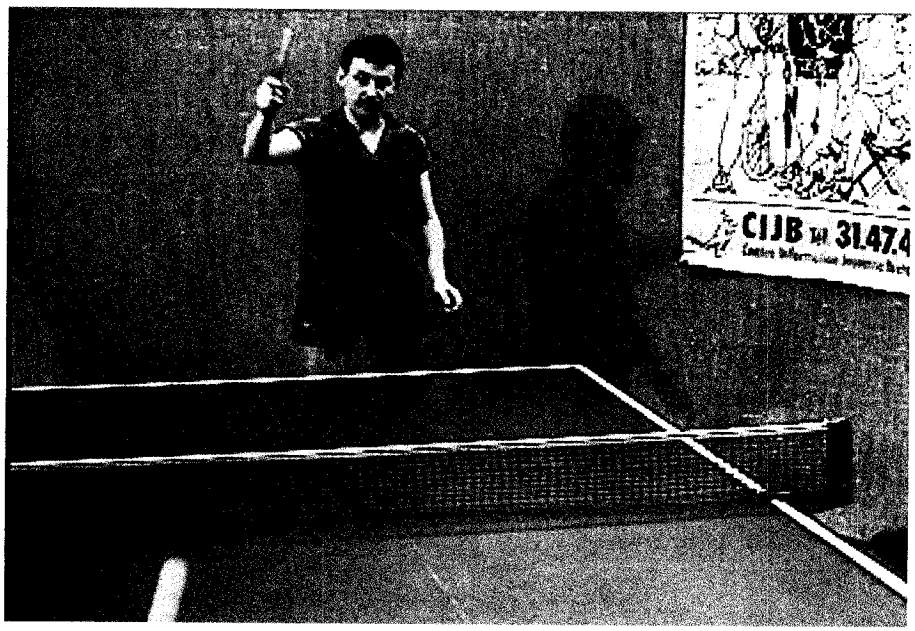

Figure 5.27: Decompressed error-free frame 83 of video sequence "Table-Tennis"

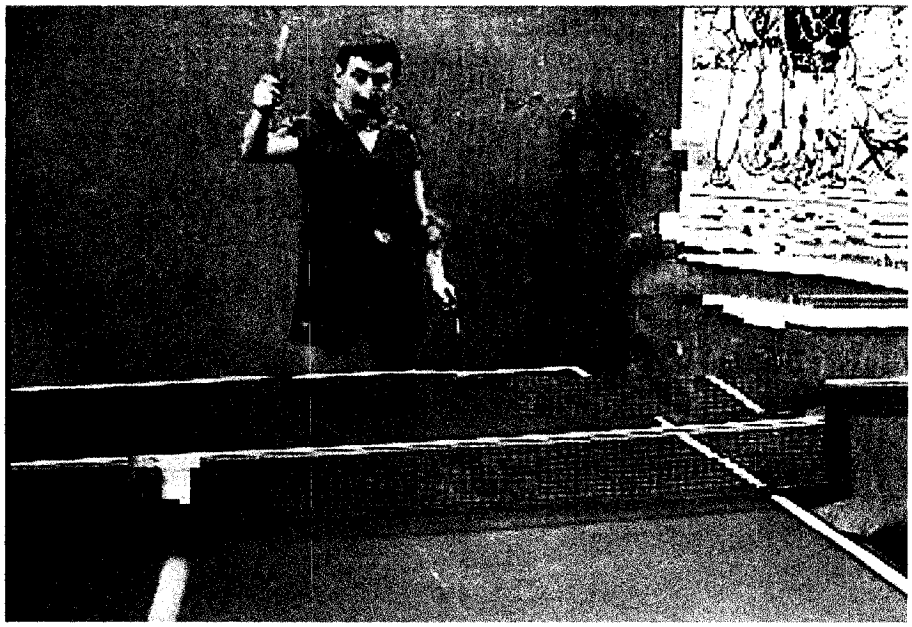

Figure 5.28: Frame 83 of video sequence "Table-Tennis" transmitted over AWGN channel at 2.3dB SNR and decoded using convolutional decoding

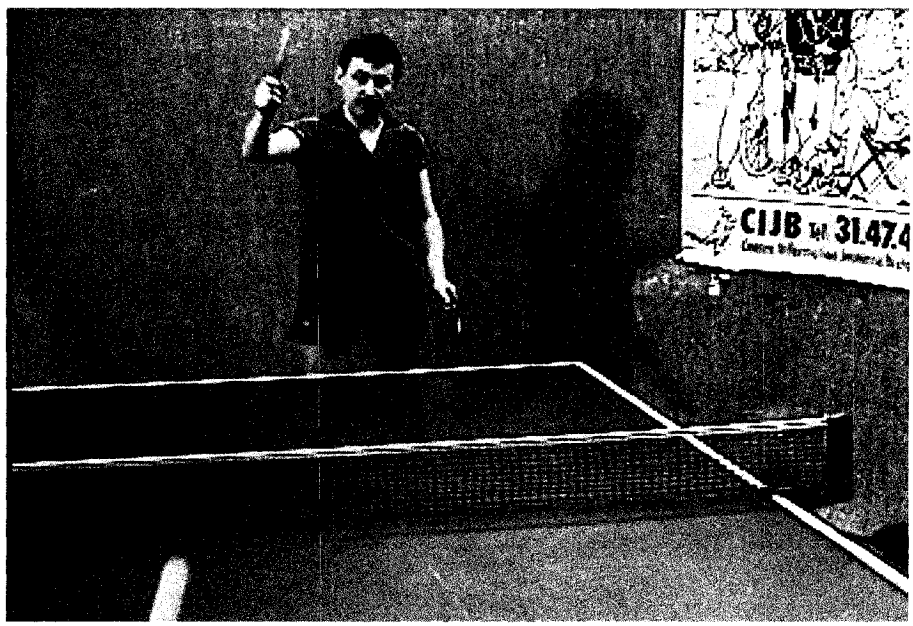

Figure 5.29: Frame 83 of video sequence "Table-Tennis" transmitted over AWGN channel at 2.3dB SNR and decoded using IJSCD with 1 iteration

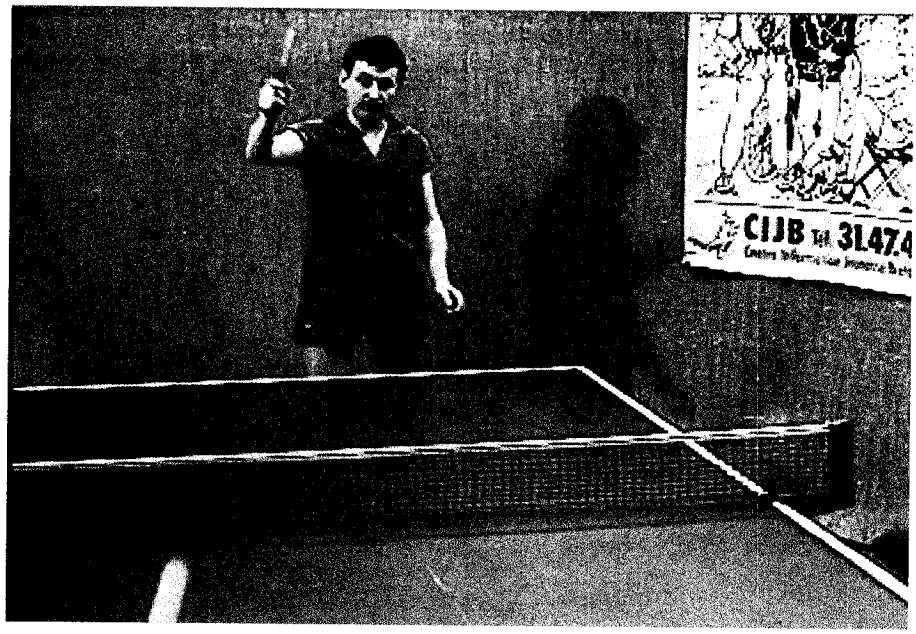

Figure 5.30: Frame 83 of video sequence "Table-Tennis" transmitted over AWGN channel at 2.3dB SNR and decoded using IJSCD with 2 iterations

5.2.4 Complexity of IJSCD

The complexity of the proposed IJSCD scheme is evaluated for one iteration using the time ratio (described in Section 4.3.4) as a measure. The time ratio for one IJSCD iteration is the sum of the time ratios of the Parity Splitter, the SOVA Decoder, the Deinterleaver, the Stream Merger, the SCG, the SSV, the Modifier and the Interleaver (on the final iteration, the Modifier and Interleaver are not used).

An experiment was performed to observe the time ratio for each of the modules. The video sequence "Football" was used under the same conditions as in the objective and subjective performance experiments. The experiment consisted of running the IJSCD scheme 5 times with independent seeds at several channel SNR values. In each run, the total run time of each module was observed and divided by the total run time of the decompressor. The average of the 5 runs was taken.

Figure 5.31 shows the results of the experiment for several channel SNR values between 1.8dB and 2.5dB. It is observed that the time ratio increases as the channel SNR decreases. This is because a lower channel SNR results in more errors, which causes the SSV to checks the semantics of more slice candidates (as observed in Section 4.3.4.1).

With the exception of the SSV, the complexity of all other modules is not dependent on the channel SNR.

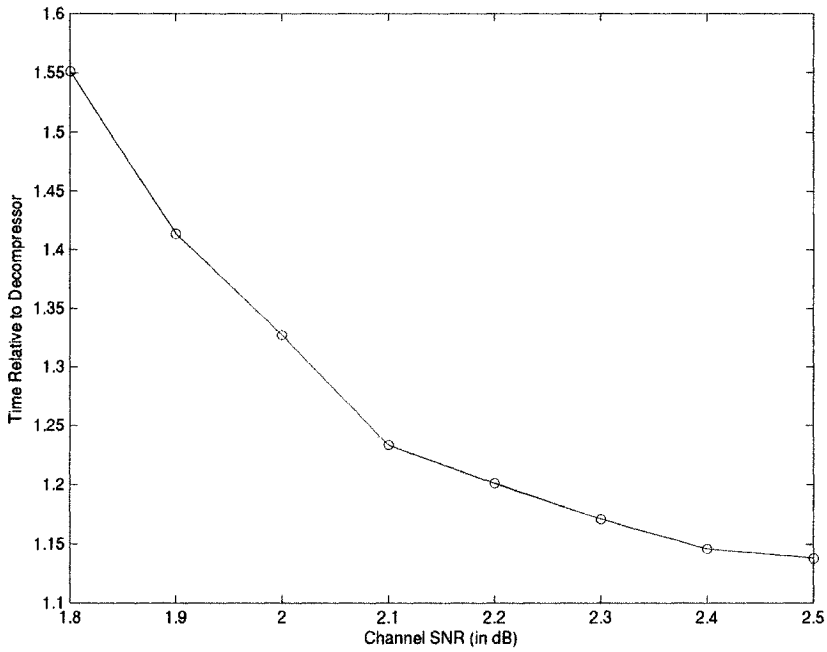

Figure 5.31: Time ratio between one IJSCD iteration and the H.264 decompressor for video sequence "Football"

It is observed that, for channel SNR values above 2.2dB, one IJSCD iteration takes less than 20% more time than a full H.264 decompression. As the channel SNR decreases, the time ratio increases. At 1.8dB channel SNR, one IJSCD iteration takes approximately 55% more time than a full H.264 decompression.

Table 5.11 gives a breakdown of the time ratio for each of the modules in one IJSCD iteration at 2.3dB channel SNR. Table 5.11 also shows the percentage of the iteration time that each module takes. Over 80% of the iteration time is spent between the SOVA Decoder and the SSV while the Parity Splitter, the Deinterleaver, the Stream Merger and the Interleaver combined take up less than 3 percent of the iteration time.

Module	Time Ratio	% of Iteration Time
Parity Splitter	0.0002	0.017%
SOVA Decoder	0.4744	40.06%
Deinterleaver	0.0232	1.96%
Stream Merger	0.0003	0.025%
SCG	0.1289	10.88%
SSV	0.4771	40.29%
Modifier	0.0573	4.84%
Interleaver	0.0229	1.93%
TOTAL	1.1843	100%

Table 5.11: Time ratio between each module in the IJSCD scheme and the H.264 Decompressor for video "Football" at 2.3dB channel SNR

Each IJSCD runs with approximately the same complexity. On the last IJSCD iteration, the Modifier and Interleaver are not run and the video sequence is decompressed. The time ratio of the last iteration is therefore 0.932 greater than the time ratio for all other iterations (the time ratio of the decompressor is always 1). Thus, the overall time ratio for N IJSCD iterations at 2.3dB channel SNR is approximately $1.18N + 0.932$. This means that when 1 iteration is used, the IJSCD scheme takes slightly more than double the time of a full H.264 decompression. When 2 iterations are used, the scheme takes slightly more than triple the time of a full H.264 decompression, and so on.

5.2.5 Comparison to IJSCDTC

In this section, the proposed IJSCD scheme is compared to IJSCDTC, the scheme proposed in [39]. First, several differences between IJSCD and IJSCDTC are highlighted. Then, the performances of the schemes are compared.

5.2.5.1 Differences Between IJSCD and IJSCDTC

IJSCDTC operates on VLC encoded compressed video while IJSCD operates on compressed video using CABAC entropy coding. It was seen that CABAC is a more efficient entropy code than VLC and it leads to a high Error Detection Probability.

IJSCD also uses a better base concealment than IJSCDTC. IJSCD resorts to temporal replacement when it cannot correct an error. On the other hand, IJSCDTC fills in all

missing pixels with black. Thus, the base concealment in IJSCDTC creates a larger contrast between error-free frames and erroneous frames than the base concealment scheme in IJSCD.

Furthermore, IJSCD uses one slice per row of macroblocks, whereas IJSCDTC uses four slices per row of macroblocks. Because each frame in IJSCDTC has more slices, more overhead bits are used.

The SCG used in IJSCD outperforms its equivalent module in IJSCDTC. In IJSCDTC, only 4 or 6 bits can possibly be flip-bits. The slice candidates in IJSCDTC are generated by flipping some or all of the possible flip-bits. Because IJSCDTC limits the possible flip-bits, it may miss out on slice candidates that are more likely than some of the ones it generates. In IJSCD, any bit can be a flip-bit. Thus, the n_{sc} slice candidates produced are actually the n_{sc} most likely candidates.

5.2.5.2 Performance Comparison Between IJSCD and IJSCDTC

When comparing the performance of IJSCD to the performance of IJSCDTC, bandwidth expansion, operating channel SNR, complexity and gain in channel SNR over the base scheme are considered.

The base scheme used for IJSCDTC is Turbo Decoding with a rate-½ turbo code. The base scheme used for IJSCD is convolutional decoding using a rate-½ convolutional code. Since IJSCDTC and IJSCD use channel codes with the same rate, they have roughly the same bandwidth expansion.

IJSCDTC operates at a lower channel SNR than IJSCD. However, when 2 or more IJSCD iterations are used, the gain provided by IJSCDTC over IJSCD is not large. On the other hand, IJSCD is significantly less complex than IJSCDTC.

IJSCDTC runs for 15 iterations, with each iteration taking about 1.5 times as much time as one turbo decoder iteration. A turbo decoder iteration consists of two BCJR decoders [8], which are more complex than SOVA Decoders [8]. Thus, 15 iterations IJSCDTC, is over 45 times as complex as a SOVA Decoder. In contrast, one IJSCD iteration is about 2.5 times as complex as a SOVA Decoder (calculated from Table 5.11). When IJSCD uses 2 iterations, it is approximately 5 times as complex as a SOVA

Decoder. Thus IJSCD using 2 iterations is more than 9 times less complex than IJSCDTC.

In addition to being less complex than IJSCDTC, IJSCD was observed to provide a larger gain over its base scheme.

5.3 Implementation Considerations

Section 4.4 discussed the way ECSS could be implemented such that it could run as either a streaming or an interactive process. In the implementation, several parts of ECSS would either be parallelized or pipelined. A similar approach could be taken for IJSCD. As the IJSCD decoder includes the modules of the ECSS decoder, the parallelization and pipelining methods of Section 4.4 could be applied to IJSCD. In addition, the SOVA decoder and Modifier could also be parallelized.

The SOVA Decoder could be parallelized because each information block generated by the interleaver produces 5 convolutional code blocks. Each convolutional code block could be decoded by a separate SOVA Decoder. This would reduce the run time of the SOVA Decoder by a factor of 5.

Once a winning slice candidate has been selected for each slice in a frame, separate Modifiers could be used to alter the soft values for each slice. In addition, each individual soft bit value could be altered by its own Modifier. Thus, the entire modification process for a frame could be reduced to a single operation.

While it may be feasible to implement IJSCD as a streaming process, implementing IJSCD as an interactive process presents several challenges. In IJSCD, the Channel Decoder must wait for the results from one iteration before proceeding to the next. Furthermore, the Interleaver used in IJSCD operates on 50000-bit information blocks. In some cases, the bits in an information block span multiple frames. When this happens, the Source Decoder must wait for the next information block before it can proceed, thus resulting in a delay. This delay could be resolved by changing the interleaver. A new interleaver could be designed such that each information block spans exactly one frame. Each information block would then be split into an appropriate number of convolutional code blocks.

5.4 Summary

This chapter presented a transmission scheme for H.264 video that uses both source and channel coding. The proposed scheme, IJSCD, builds on ECSS from Chapter 4 in using soft channel information to aid source decoding and proposes a method of using source information to aid channel decoding. A feedback loop is thus created between the source and channel decoders, which allows decoding to be performed in an iterative manner. Source information is fed into the channel decoder by modifying the channel decoded soft bit values.

The objective and subjective performance results show that, for the same bandwidth requirements, a single iteration of the proposed scheme significantly outperforms convolutional decoding alone. The results also show that performing additional iterations further increases performance.

The complexity of the proposed scheme was determined for a single iteration. The total complexity of the scheme is a function of the number of iterations performed. Thus a tradeoff can be made between complexity and performance.

Chapter 6

Conclusion

6.1 Contributions

This thesis presents an approach to error resilient transmission of H.264 compressed video using source semantics. CABAC entropy coding was used. Several experiments were first performed to observe how bit errors affect H.264 decompression. Two transmission schemes were then proposed. This has lead to several contributions.

Observations were made as to how semantic errors are detected in H.264 using CABAC entropy coding. It was seen that bit errors cause semantic errors over 99% of the time. It was also seen that bit errors are more likely to be detected after short delays than after long delays, but that long delays are not uncommon. Finally, it was seen that undetected bit errors do not seriously affect video quality.

Error Concealment using Source Semantics (ECSS), the first scheme proposed, improves the performance of video transmission without using a channel code. Thus, no bandwidth expansion occurs. In the scheme, soft channel information is used to select and rank a set of slice candidates in descending order of likelihood. The ranking of slice candidates is implemented using the Incomplete Partial Sums Algorithm (IPSA).

Iterative Joint Source Channel Decoding (IJSCD), the second scheme proposed, combines source and channel decoding in an iterative manner. A convolutional code is used as the channel code and soft decoding is used to provide the source decoder with soft information. After source decoding, information is fed back to the channel decoder using a modification parameter. This allows the source and channel decoders to interact in an iterative manner.

6.2 Conclusions

Based on the research presented in this thesis, several conclusions can be drawn.

- Semantic correctness is a good indicator of slice correctness for H.264 compressed video using CABAC entropy coding and can be used in a decoding scheme.
- There is potential to remove redundancy in intra prediction, as most semantic errors are invalid intra prediction mode errors. A different way of describing prediction modes could make it impossible to have invalid prediction modes.

However, the ability to detect invalid prediction modes is used as an advantage in the schemes proposed in this thesis.

- By feeding source information back into the channel decoder, additional errors can be corrected. Thus, iterative decoding improves performance as compared to non-iterative decoding.
- The detection location of a slice candidate can be used as a measure for comparing slice candidates. When deciding between two slice candidates, the candidate with the later detection location yields a better objective performance.
- Semantic verification fails to improve performance when there are too many bit errors in a slice. This is because soft channel information is not very reliable when the channel SNR is low.

6.3 Future Work

The schemes proposed in this thesis were tested for H.264 compressed video using CABAC entropy coding. In theory, the same approach can be taken when VLC entropy coding is used, however several parameters may need to be changed for decoding to be optimized.

Currently, the maximum number of slice candidates to be verified by the SSV is fixed and must be decided upon before a video sequence is received. This could be modified so that the maximum number of slice candidates is dynamic and a function of the desired complexity. Slices with few to no errors would thus allow more time to be allotted to a slice with many errors. In this way, the erroneous slice has more chances to be corrected. Furthermore, the number of slice candidates generated can change between iterations.

Another potential improvement is to use information about the slice candidates that were rejected by the SSV when the Modifier alters the soft bit values. The rejected slice candidates may give an indication about the correctness or incorrectness of several bit decisions.

The channel code used in IJSCD is a convolutional code. The turbo dynamic of the IJSCD decoder is between the source decoder and the channel decoder. In the future, the scheme could be implemented using an actual turbo code for the channel code. Rather

than being part of the turbo dynamic, the source decoder would then be used to modify information between iterations of the turbo decoder.

The method by which the Modifier feeds source information back to the channel decoder was developed in an ad-hoc manner. While some theoretical work is done, there are still several assumptions made. In the end, the formula for modification depends on a parameter that is determined empirically. In the future, more theoretical work could be done to develop a purely mathematical approach to feeding source information into a channel decoder.

REFERENCES

[1] Y. Wang, J. Ostermann and Y. Zhang, *Video Processing and Communications*. Upper Saddle River, NJ: Prentice Hall, 2002.

[2] *The official MPEG website, http://www.chiariglione.org/MPEG/.*

[3] S. Aramvith and M. Sun, "MPEG-1 and MPEG-2 video standards," *Handbook of Video and Image Processing*, A. Bovik, ed., Academic Press, San Diego, CA, 2000, pp. 597-611.

[4] T. Ebrahimi and C. Horne, "MPEG-4 natural video coding – An overview," *Signal Processing, Image Communication*, vol. 15, no. 4-5, pp. 365-385, 2000.

[5] S. Kwon, A. Tamhankar, K. R. Rao, "Overview of H.264/MPEG-4 Part 10," *Journal of Visual Communication and Image Representation*, vol. 17, issue 2, April 2006, pp. 186-216.

[6] Y. Wang and Q. F. Zhu, "Error control and concealment for video communications: a review," *Proceedings of the IEEE*, vol. 86, May 1998, pp. 974-997.

[7] J. G. Proakis, *Digital Communications*. New York, NY: McGraw Hill, 4th ed., 2001.

[8] S. Lin and D. J. Costello Jr., *Error Control Coding: Fundamentals and Applications*. Upper Saddle River, NJ: Prentice Hall, 2nd ed., 2004.

[9] Joint Video Team (JVT), "Recommendation ITU-T H.264: Advanced Video Coding for Generic Audiovisual Services," ITU-T, May 2003.

[10] A. Puri, X. Chen and A. Luthra, "Video coding using the H.264/MPEG-4 AVC compression standard," *Signal Processing, Image Communication*, vol. 19, no. 9, October 2004, pp. 793-849.

[11] G. J. Sullivan, P. Topiwala and A. Luthra, "The H.264/AVC Advanced Video Coding standard: Overview and introduction to the Fidelity Range Extensions," *SPIE Conference on Applications of Digital Image Processing XXVII*, vol. 5558, pp. 53-74, August 2004.

[12] I. E. G. Richardson, *H.264 and MPEG-4 Video Compression*. The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England: John Wiley & Sons, 1st ed., 2003.

[13] D. Marpe, H. Schwarz and T. Wiegand, "Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, issue 7, July 2003, pp. 620-636.

[14] D. Marpe, T. Wiegand and G. J. Sullivan, "The H.264/MPEG-4 advanced video coding standard and its applications," *IEEE Communications Magazine*, vol. 44, issue 8, 2006, pp. 134-143.

[15] T. Wiegand, G. J. Sullivan, G. Bjntegaard and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, issue 7, July 2003, pp. 560-576.

[16] B. Sklar, *Digital Communications: Fundamentals and Applications*. Upper Saddle River, New Jersey: Prentice Hall, 2nd ed., 2001.

[17] Q. Chen and K. P. Subbalakshmi, "Joint source-channel decoding for MPEG-4 video transmission over wireless channels," *IEEE Journal on Selected Areas in Communications*, vol. 21, issue 10, December 2003, pp. 1780-1789.

[18] H. Nguyen and P. Duhamel, "Robust source decoding of variable-length encoded video data taking into account source constraints," *IEEE Transactions on Communications*, vol. 53, issue 7, 2005, pp. 1077-1084.

[19] M. Bystrom, S. Kaiser and A. Kopansky, "Soft source decoding with applications," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, issue 10, 2001, pp. 1108-1120.

[20] M. Jeanne, J. C. Carlach, P. Siohan and L. Guivarch, "Source and joint source-channel decoding of variable length codes," *International Conference on Communications*, vol. 2, 2002, pp.768-772.

[21] Y. Wang and S. Yu, "Joint source-channel decoding for H.264 coded video stream," *IEEE Transactions on Consumer Electronics*, vol. 51, issue 4, 2005, pp. 1273-1276.

[22] A. H. Murad, T. E. Fuja, "Joint source-channel decoding of variable length encoded sources", *IEEE Information Theory Workshop*, Killarney, Ireland, June 1998, pp.1-5.

[23] L. Pu, Z. Wu, A. Bilgin, M. Marcellin, B. Vasic, "LDPC-based iterative joint source-channel decoding scheme for JPEG2000," *IEEE Transactions on Image Processing*, vol. 16, issue 2, 2007, pp. 577-581.

[24] Z. Peng, Y. Huang and D. Costello, "Turbo codes for image transmission - a joint channel and source decoding approach," *IEEE Journal on Selected Areas in Communications*, vol. 18, issue 6, 2000, pp. 868-879.

[25] Z. Peng, Y. Huang, D. Costello and R. L. Stevenson, "Joint channel and source decoding for vector quantized images using turbo codes," *ISCAS 1998*, vol. 4, May-June 1998, pp. 5-8.

[26] Z. Peng, Y. Huang, D. Costello and R. L. Stevenson, "Joint channel and source decoding for subband coded image," *ICIP 1998*, vol. 1, October 1998, pp. 329-333.

[27] T. P. Chen and T. Chen, "Second generation error concealment for video transport over error prone channels," *ICIP 2002*, vol. 1, September 2002, pp. 25-28.

[28] Y. Wang, Q. Zhu and L. Shaw, "Coding and cell-loss recovery in DCT based packet video," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 3, issue 3, June, 1993, pp. 248-258.

[29] S. Hemami and T. H. Y. Meng, "Transform coded image reconstruction exploiting interblock correlation," *IEEE Transactions on Image Processing*, vol. 4, issue 7, July 1995, pp. 1023-1027.

[30] J. W. Suh and Y. S. Ho, "Error concealment based on directional interpolation," *IEEE Transactions on Consumer Electronics*, vol. 43, issue 3, August 1997, pp. 295-302.

[31] W. Zhu, Y. Wang and Q. F. Zhu, "Second-order derivative-based smoothness measure for error concealment in DCT-based codecs," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 8, issue 6, October 1998, pp. 713-718.

[32] W. M. Lam, A. Reibman and B. Liu, "Recovery of lost or erroneously received motion vectors," *ICASSP 1993*, vol. 5, April 1993, pp. 417-420.

[33] J. Zhang, J. F. Arnold, M. R. Frater and M. R. Pickering, "Video error concealment using decoder motion vector estimation," *TENCON 1997*, vol. 2, December 1997, pp. 777-780

[34] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*. New York: Cambridge University Press, 1992.

[35] W. E. Lynch, V. Papadakis, R. Krishnamurthy, T. Le-Ngoc, "Syntax based error concealment," *Signal Processing: Image Communication*, vol. 16, no. 9, June 2001, pp. 827–835.

[36] W. E. Lynch, V. Papadakis, R. Krishnamurthy and T. Le-Ngoc, "Syntax and discontinuity based error concealment," *ISCAS 1999*, vol. 4, May 1999, pp. 235-238.

[37] Y. Mei, T. Le-Ngoc, and W. E. Lynch, "A combined multiple candidate likelihood decoding and error concealment scheme for compressed video transmission over noisy channels," *Signal Processing: Image Communication*, vol. 18, no. 10, October 2003, pp. 971–980.

[38] X. F. Ma, *Iterative Joint Source and Channel Decoding Using Turbo Codes for MPEG-4 Video Transmission*, master's thesis, Concordia University, Dept. of Electrical and Computer Engineering, 2004.

[39] X. F. Ma, W. E. Lynch, "Iterative joint source-channel decoding using Turbo Codes for MPEG-4 video transmission," *International Conference on Acoustics, Speech and Signal Processing*, vol. 4, 2004, pp. 657-660.

[40] J. Hagenauer and P. Hoeher, "A Viterbi Algorithm with soft-decision outputs and its applications," *Global Telecommunications Conference*, vol. 3, 1989, pp. 1680-1686.

[41] G. D. Forney, "The Viterbi Algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, March 1973, pp. 268-278.

[42] D. Levine, W. Lynch and T. Le-Ngoc, "Iterative joint source-channel decoding of H.264 compressed video," *ISCAS 2007*, May 2007, pp. 1517-1520.

[43] D. Levine, W. Lynch and T. Le-Ngoc, "Observations on error detection in H.264," *MWSCAS 2007*, August 2007.