ITERATIVE JOINT SOURCE CHANNEL DECODING FOR H.264 COMPRESSED VIDEO TRANSMISSION

Nguyen Nguyen Quang

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

April 2010

Your file Votre référence
ISBN: 978-0-494-67164-1
Our file Notre référence
ISBN: 978-0-494-67164-1

ABSTRACT

Iterative Joint Source Channel Decoding for H.264 Compressed Video Transmission

Nguyen Nguyen Quang

In this thesis, the error resilient transmission of H.264 compressed video using Context-based Adaptive Binary Arithmetic Code (CABAC) as the entropy code is examined. The H.264 compressed video is convolutionally encoded and transmitted over an Additive White Gaussian Noise (AWGN) channel. Two iterative joint source-channel decoding schemes are proposed, in which slice candidates that failed semantic verification are exploited. The first proposed scheme uses soft values of bits produced by a soft-input soft-output channel decoder to generate a list of slice candidates for each slice in the compressed video sequence. These slice candidates are semantically verified to choose the best one. A new semantic checking method is proposed, which uses information from slice candidates that failed semantic verification to virtually check the current slice candidate. The second proposed scheme is built on the first one. This scheme also uses slice candidates that failed semantic verification but it uses them to modify soft values of bits at the source decoder before they are fed back into the channel decoder for the next iteration. Simulation results show that both schemes offer improvements in terms of subjective quality and in terms of objective quality using PSNR and BER as measures.

Keywords: Video transmission, H.264, semantics, slice candidate, joint source-channel decoding, error resiliency.

To my family.

ACKNOWLEDGEMENTS

First I would like to express my deepest gratitude and respect to my two supervisors Dr. William E. Lynch and Dr. Tho Le-Ngoc for proposing an interesting research topic, for providing continuous support and for directing my research. I would also like to thank all professors with whom I have interacted during my studies at Concordia University. In addition, I would like to thank David Levine for his kindly help.

Second I would like to thank my parents. Their belief in me helps me overcome difficulties during my research.

Finally, thanks to all my friends at Concordia University for allowing me to share my experience in study and research with them.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AC	Arithmetic Code
APP	A-*Posteriori* Probability
AVC	Advanced Video Coding
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BPSK	Binary Phase-Shift Keying
BSC	Best Slice Candidate
CABAC	Context-based Adaptive Binary Arithmetic Coding
CAVLC	Context Adaptive Variable Length Coding
CDF	Cumulative Density Function
DB	Detection Bit
DVD	Digital Versatile Disc
EDD	Error Detection Delay
EOS	End Of Slice
FS	Fast Search
GOP	Group Of Pictures
ICT	Integer Cosine Transform
IJSCD-VC	Iterative Joint Source-Channel Decoding using Virtual Checking Method
IJSCD-VVC	Iterative Joint Source-Channel Decoding using Voting and Virtual Checking Method

JSCD	Joint Source Channel Decoding
JVT	Joint Video Team
MAP	Maximum *A-Posteriori*
MB	Macroblock
MPEG	Motion Picture Expert Group
PI	Performance Improvement
PSC	Primary Slice Candidate
PSNR	Peak Signal-to-Noise Ratio
RSC	Recursive Systematic Convolutional
SCG	Slice Candidate Generator
SOS	Start Of Slice
SPS	Sequence Parameter Set
SSV	Source Semantic Verifier
SVC	Scalable Video Coding
VC	Virtual Checking
VCEG	Video Coding Experts Group
VLC	Variable Length Code
VVC	Voting and Virtual Checking

LIST OF SYMBOLS

$\boldsymbol{u}$	Information bit sequence
$\boldsymbol{y}$	Received noisy bit sequence after the channel
$L(u_k \vert y)$	Soft value of bit u_k obtained at the MAP decoder
s', s	State of the RSC encoder
$\alpha_k(s)$	Forward metric
$\beta_k(s)$	Backward metric
$\gamma_k(s',s)$	Branch metric
$P(v)$	The likelihood of the PSC
N_{SC}	Number of slice candidates actually verified
N_{MAX}	Maximum number of slice candidates virtually verified
λ_k	Total number of votes $u_k = 1$
$\psi_{k,m}$	A vote for bit u_k given by slice candidate *m*
$L(u_k \vert \lambda_k)$	Source intrinsic information of bit u_k
$L'(u_k \vert y_{[j]})$	Modified soft value of bit u_k at the j^th iteration when the BSC passes verification
$L'(u_k \vert y_{[j]}, \lambda_k)$	Modified soft value of bit u_k at the j^th iteration when the BSC fails verification

Chapter 1
Introduction

1.1 Transmission of Compressed Video

In recent years, there has been an increasing demand for video services over a variety of wireless channels. However, transmitting video signals requires a large amount of resources. For example, consider a one-second video sequence that has a frame rate of 30 frames/second, a frame size of 352x240 pixels, and uses YUV 4:4:4 sampling. To transmit this video sequence, the required bandwidth is about 60Mp/s, which is too large in practice. Therefore, video compression, a process of compacting video data into a smaller number of bits by exploiting the spatial and temporal redundancy of the video sequence is used to reduce the required bandwidth [1]. In addition to transmission, video compression is also important in video storage since it increases the quantity of video data that can be stored.

Several video compression standards have been developed to address the issue of video transmission and storage. For example, the Motion Picture Expert Group (MPEG) has developed several video compression standards such as MPEG-1, MPEG-2 and MPEG-4 that have been internationally accepted. These video standards have led to a wide range of applications such as Digital Versatile Disc (DVD), Blu-ray Disc, High Definition TV (HDTV), Internet video streaming, and videoconferencing.

More recently, H.264/AVC video compression standard has been developed and been widely accepted. This new video compression standard provides higher compression performance than previous standards [2] and better support for reliable transmission [1]. H.264 technology is currently used in Blu-ray Disc, HDTV broadcasting, and a variety of mobile devices. In this thesis, the transmission of H.264 compressed video is studied.

When transmitted over communication channels, a compressed video may suffer from channel noise and hence bit errors occur. There are several types of transmission errors that can occur in a noisy channel including erasures caused by packet loss in network congestion, burst errors caused by multipath fading channels and random bit errors caused by random channel noise. In this thesis, an Additive White Gaussian Noise (AWGN) channel is assumed and thus only random bit errors are considered.

Since compression removes spatial and temporal redundancy in a video sequence, it makes a compressed video more fragile to transmission errors than an uncompressed video. Due to the compression mechanisms in H.264, a single bit error can propagate both within a frame and between frames, which is called the error propagation effect. One mechanism that causes the error propagation effect is the use of entropy coding. Another cause of error propagation is the use of spatial-temporal prediction in video coding. Thus, a single bit error can propagate and cause a large portion of video to be corrupted. Therefore the transmission of compressed video sequences over wireless channels is a challenging task.

1.2 Problem Statement

In order to enhance error resilience of compressed video when transmitted over wireless channels, many approaches have been proposed [3]. In general, the most common approach to enhance error resilience is the use of channel coding, which systematically adds redundant information (i.e. parity check bits) into a bit stream to correct transmission errors. Another approach is the use of residual redundancy in the source coding. In particular, after compression, a compressed video sequence still contains residual redundancy, and this redundancy information can be exploited to correct transmission errors without any extra information being added into a bit stream. Furthermore, these two approaches can be combined so that the channel decoder and the source decoder can jointly correct transmission errors.

This thesis investigates methods of improving error resilience in the transmission of H.264 compressed videos that use CABAC as the entropy code. Particularly, both the redundancy systematically added by the channel coding and the residual source redundancy in a compressed video are exploited, in a co-operative manner, to detect and correct transmission errors. The wireless channel considered in the thesis is an AWGN channel.

1.3 Figures of Merit

In order to evaluate the performance of the schemes proposed in the thesis, several figures of merit are used. In particular, the figures used are the objective quality (including Peak Signal to Noise Ratio (PSNR) and Bit Error Rate (BER)), the subjective quality, and the computational complexity.

PSNR is the most commonly used measure for the quality of a reconstructed video sequence when compared to the original video sequence. It is measured on a logarithmic scale and depends on the mean squared error of between the original frame and the reconstructed frame [1]. The PSNR of a video sequence is measured for the luminance (Y), blue chrominance (U) and red chrominance (V) components, which are respectively defined as follows:

$$PSNR(Y) = 10\log_{10}\frac{255^2}{\frac{1}{N_Y}\sum_{i=1}^{N_Y}(p_{oY}[i]-p_{rY}[i])^2} \tag{1.1}$$

$$PSNR(U) = 10\log_{10}\frac{255^2}{\frac{1}{N_U}\sum_{i=1}^{N_U}(p_{oU}[i]-p_{rU}[i])^2} \tag{1.2}$$

$$PSNR(V) = 10\log_{10}\frac{255^2}{\frac{1}{N_V}\sum_{i=1}^{N_V}(p_{oV}[i]-p_{rV}[i])^2} \tag{1.3}$$

Here, N_Y , N_U , N_V are the number of luminance, blue chrominance and red chrominance pixels in a frame, respectively. $p_{oY}[i]$, $p_{oU}[i]$, $p_{oV}[i]$ are the luminance, blue chrominance and red chrominance pixel values in the original frame. $p_{rY}[i]$, $p_{rU}[i]$, $p_{rV}[i]$ are the luminance, blue chrominance and red chrominance pixel values in the reconstructed frame.

Also, BER is a measure used to evaluate the quality of a reconstructed bit stream (from the viewpoint of channel coding). It is calculated as the number of bit errors divided by the total number of bits in the bit stream, as shown in Equation 1.4.

$$BER = \frac{\text{number of error bits}}{\text{number of total bits}} \tag{1.4}$$

For example, a bit stream of length 10000 bits is transmitted over a noisy channel. At the receiver side, the received bit stream has 5 bit errors. Thus, BER in this case is 0.0005.

In this thesis, BER of the compressed video bit stream obtained after the channel decoder and BER of the compressed video bit stream obtained after the proposed error correction schemes have been performed are shown for comparison.

For subjective video quality, rigorous subjective quality testing is not performed due to the consideration of cost and time. Instead, several frames of the original and reconstructed video are shown for reader's own evaluation and commented by the author.

The computational complexity of the proposed schemes in this thesis is also a figure of merit. The complexity is measured by "the number of slice candidates verified" and by the run time (measured in seconds). In addition, the time ratio of a module, which is defined as the run time of that module to the run time of the H.264 decompressor, is also used as the measure of computational complexity.

1.4 Thesis Outline

This thesis is organized as follows:

Chapter 2 reviews the background about the H.264/AVC video coding standard including some important coding features of the H.264/AVC, the H.264 compressor and decompressor. A literature review on joint source-channel decoding for the transmission of images and videos is also presented.

Chapter 3 presents the first proposed scheme: Iterative Joint Source-Channel Decoding using Virtual Checking method (IJSCD-VC). The scheme makes use of the H.264 source semantics residual redundancy together with soft values of bits produced by the channel decoder to correct transmission errors in an iterative manner. A method, called Virtual Checking, is proposed, which uses information of slice candidates that failed semantic verification to virtually check the current slice candidate. Also, this scheme follows a previous work to modify soft values of bits before feeding them back into the channel decoder for the next iteration.

Chapter 4 presents the second proposed scheme: Iterative Joint Source-Channel Decoding using Voting and Virtual Checking method (IJSCD-VVC), which is built on the first scheme. A new method of modifying soft values of bits at the source decoder is proposed, namely the Voting method, which makes use of information of several slice candidates that failed semantic verification.

Chapter 5 concludes the thesis and discusses some future work.

Chapter 2
Background and Literature Review

In this chapter, background information on video compression and decompression using the H.264/AVC standard is covered. In particular, the structure of the H.264 bit stream as well as mechanisms for error propagation are discussed.

Like previous video coding standards (the MPEG-4 standard for example), the H.264 compressed bit stream is structured in such a way that error resilience is enhanced, which makes it suitable for use in error-prone environments.

As discussed in Chapter 1, a bit error often propagates both within a frame and between frames in a compressed video sequence. Thus, this chapter also explains the operation of the H.264 compressor and decompressor to clarify the mechanisms that cause errors to propagate in the compressed video bit stream as well as to shed light on the question of how errors are detected by the H.264 decompressor during the decompression.

Section 2.1 discusses the structure of the H.264 compressed bit stream and explains the operation of the H.264 compressor and decompressor. Section 2.2 presents a literature review of existing joint source-channel decoding schemes for the error resilient transmission of images and videos. Section 2.3 summarizes the chapter.

2.1 H.264 Video Compression

H.264/AVC is the standard developed by the Joint Video Team (JVT), including experts from the Video Coding Experts Group (VCEG) and the Motion Picture Expert Group (MPEG). The H.264/AVC standard is also called the MPEG-4 Part 10 standard. In this section, background information about the H.264/AVC standard relevant to the work of the thesis is provided. Section 2.1.1 represents the overview of the H.264 standard. Section 2.1.2 discusses the hierarchical structure of the H.264 compressed bit stream. Section 2.1.3 discusses video coding techniques used in the H.264 compression/decompression. Section 2.1.4 and 2.1.5 describes the H.264 compressor and decompressor respectively.

2.1.1 Overview of the H.264/AVC Standard

The H.264/AVC standard comprises a set of seven profiles [1]. Each supports a particular set of coding features and each is suitable for a particular set of applications.

- ❖ Baseline Profile: suitable for low cost application such as videoconferencing and mobile video.
- ❖ Main Profile: intended for broadcast and storage applications.
- ❖ Extended Profile: suitable for video streaming applications. This profile has relatively high compression capability and some extra tricks for robustness to data losses (e.g. data partitioning).
- ❖ High Profiles: there are four variant of High profiles, including: High Profile, High 10 Profile, High 4:2:2 Profile, High 4:4:4 Profile. These profiles are different at the maximum number of bits per pixel supported and the allowable number of chrominance samples per video frame. High Profiles are

intended for Digital Video Broadcast, high-definition television applications (Blu-Ray, HD-DVD).

Following the work in [4], [5], a video sequence is encoded using Main Profile in this thesis.

2.1.2 Structure of the H.264 bit stream

Like previous video coding standards, in the H.264/AVC standard the encoder is not standardized. Only the bit stream the encoder produces and the procedure for decoding the bit stream are standardized. The compressed bit stream generated by a compliant encoder has to follow the syntax/semantics specified by the standard. Thus, the structure of the H.264 compressed bit stream is introduced in this section.

An H.264 bit stream is hierarchically organized as different layers: Sequence, Group of Pictures (GOP), Frame, Slice, Macroblock and Block, as depicted in Figure 2.1. They are described as follows.

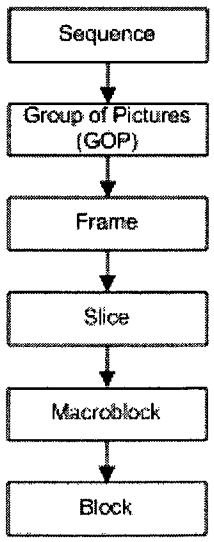


Figure 2.1: Hierarchy of an H.264 compressed bit stream.

Sequence layer

The top level in the H.264 bit stream is the sequence, which begins with a sequence header (i.e., Sequence Parameter Set (SPS)) and is followed by a series of GOPs. The SPS is a header that contains information common to the entire video sequence such as the profile, the frame size, the number of reference frames, the choice of progressive or interlaced coding [1]. An error in the sequence header can make it impossible to correctly decode the sequence. Therefore, as will be discussed in Chapter 3, bits in the slice headers and higher layers which are important for the decompression are assumed to be transmitted without errors.

GOP layer

The next level below the sequence is the GOP. A GOP is a set of adjacent frames where the number of frames and the type of each frame is specified. Each GOP starts by an I-frame and followed by P-frames and/or B-frames. Since I-frames use intra prediction (described later), for GOP in which the proportion of I-frames is low, the compression efficiency is high with the cost that there are few synchronization points which may not ideal for random access and for error resilience. Note that an access point (i.e. an I-frame) allows the decoder to start decoding properly after some loss or corruption. In contrast, for GOP in which the proportion of I-frames is high, the compression efficiency is low, however, there are more opportunities for synchronization.

Frame layer

Within a GOP are frames. There are three main types of coded frames: intra-coded frames (I-frames), predictive frames (P-frames) and bidirectional frames (B-frames). An I-frame is coded using intra prediction, i.e. prediction using pixels within the current frame only. As mentioned above, I-frames are used as the basis for decoding of other frames and provide access points to the coded sequence where decoding can begin. Meanwhile, a P-frame is coded using forward prediction, i.e. prediction using pixels from the nearest previously coded frame (i.e., a past I-frame or P-frame available in an encoder and decoder buffer). On the other hand, a B-frame is coded using bidirectional prediction, i.e. prediction using pixels from both the previous and future I-frame or P-frame.

One should note that I-frames eliminate the temporal error propagation effect in the video sequence since it is coded independently of any other coded frames. However, I-frames have poor compression efficiency since no temporal prediction is used. On the other hand, P-frames and B-frames have higher compression ratio due to temporal prediction, however, they make errors propagate.

Slice layer

The next level below the frame is the slice layer. Each frame is comprised of one or more non-overlapped slices. The H.264/AVC standard supports five types of slices: I-slice, P-slice, B-slice, SP-slice (Switching P) and SI-slice (Switching I). More details on each slice type can be found in [6]. Each slice begins with a start code (a resynchronization point). A slice header is attached after the start code, which contains the information about the frame number, the coded frame type, the slice type, the number of the first macroblock in the slice. The slice header is followed by the slice data which consists of a series of coded macroblocks. The last byte of the slice data is padded with zeros so that the slice is byte aligned [1].

As discussed in Chapter 1, a single bit error can propagate both within a frame and between frames. When an error propagates in an H.264 video sequence, the decompressed video becomes corrupted. And the video corruption continues until the decompressor reaches a resynchronization point (i.e. the start code that prefixes each slice header). Whenever the decompressor reaches a resynchronization point, it can resynchronize to the bit stream and restart decoding from the next slice header. Therefore, within a frame, the video corruption caused by loss of synchronization can not propagate beyond the end of slice. In other words, resynchronization points prevent errors from propagating beyond the end of slice. (However, one should note that error propagation still occurs between frames due to temporal prediction). In light of this, the error correction schemes proposed in this thesis are done on a slice by slice basis.

Macroblock layer

Each slice contains an integer number of macroblocks (as depicted in Figure 2.2).

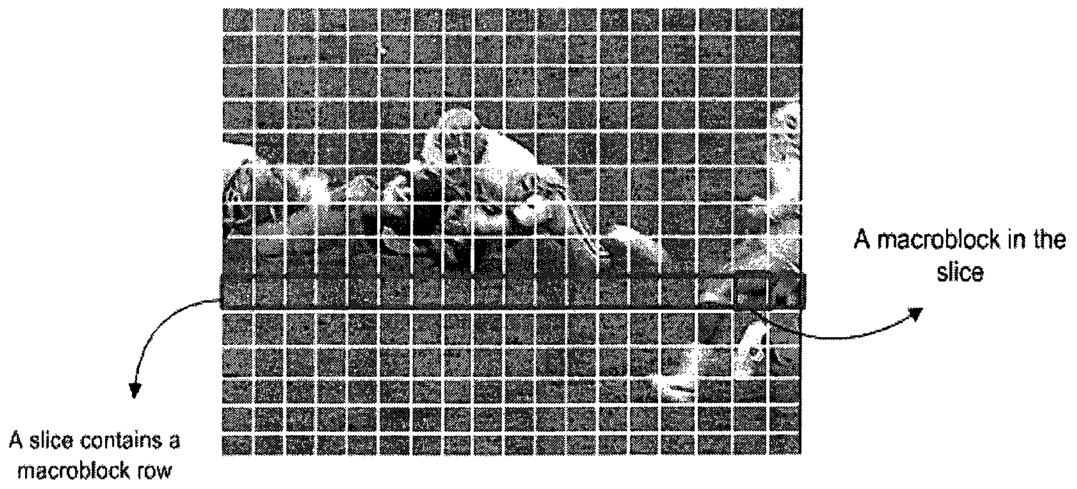


Figure 2.2: Each frame in the video comprises of non-overlapped slices. Each slice contains an integer number of macroblocks.

In the popular 4:2:0 sampling format, each macroblock contains coded data corresponding to a 16x16 luma block and two 8x8 chroma blocks. The coded data in a macroblock includes a macroblock header (describing macroblock type (I/P/B), motion vector(s)) and residual data [1].

Block layer

During the encoding process, the residual data for all macroblocks is subdivided into 4x4 blocks and is transformed using the Integer Cosine Transform (ICT).

2.1.3 Video Coding Techniques

This section discusses techniques used during the video compression/decompression process. By that, the mechanisms of error propagation effect in the compressed video bit stream are clarified. The operation of the H.264 compressor and decompressor will be discussed in Section 2.1.4 and 2.1.5 respectively.

2.1.3.1 Integer Cosine Transform (ICT)

ICT is a mathematical method that transforms image data from spatial domain to frequency domain. Using ICT, the visually important information of a block in a video frame is concentrated into a small number of coefficients which can be efficiently encoded. By this way, the amount of spatial redundancy in a residual frame can be significantly reduced [1].

2.1.3.2 Quantization

The ICT-transformed coefficients are quantized so that the near-zero coefficients are set to be zero and the remaining coefficients are represented with a reduced precision. This process causes signal loss. However, it offers better compression [1].

2.1.3.3 H.264 Prediction

In general, a coded macroblock in a slice is predicted from samples that have already been encoded. There are two types of prediction supported in H.264/AVC standard: inter prediction and intra prediction, which are to be described as follows. Note that prediction techniques used during the video compression is one cause of error propagation.

2.1.3.3.1 Inter Prediction

For inter prediction, a macroblock is predicted using information from the previously encoded frames(s). The H.264/AVC standard allows macroblocks to be partitioned for inter prediction. In particular, each 16x16 macroblock can be partitioned using either a single 16x16 partition, or two 16x8 partitions, or two 8x16 partitions, or four 8x8 partitions. When the 8x8 mode is used, each 8x8 block can be further divided into smaller blocks as either one 8x8 partition, two 8x4 partitions, two 4x8 partitions or four 4x4 partitions, as depicted in Figure 2.3.

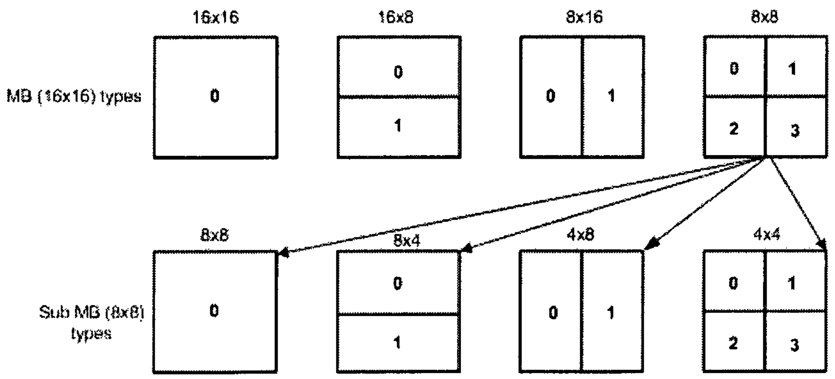


Figure 2.3: Segmentation of macroblocks for inter prediction. Top: segmentation of a 16x16 macroblock. Bottom: segmentation of an 8x8 partition [1].

Once the partition is chosen, a block matching algorithm is applied to find the best match of the partition from previously encoded frame(s). This process is called the *motion estimation*. The position of the matching block is presented by a vector called the motion vector. The selected best match block is subtracted from the original block to produce a residual block for further processing. This process is called the *motion compensation*. One should note that the choice of the partition size has an influence on the compression performance. The smaller the partition size is, the more accurate the motion estimation is, and thus the less the residual energy remains. This comes up with the cost of increased computational complexity (more search operations are required) and extra bits required to represent the motion vectors [1].

One should also note that, due to the mechanism of inter prediction, bit errors can temporally propagate between frames. Specifically, an erroneously decoded portion of an I-frame or P-frame can temporally propagate to other P-frames and/or B-frames if it is used as a reference sample for the currently decoded frame [3]. Moreover, since a B-frame can make use of reference frames before or after it in temporal order, errors can propagate forward or backward in time in a compressed video sequence.

2.1.3.3.2 Intra Prediction

For intra prediction, a macroblock is predicted using spatially neighboring blocks without using any information from other frames. To perform intra prediction, each macroblock is partitioned as either 16 4x4 blocks or one 16x16 block. For the prediction of a 4x4 block, there are nine possible prediction modes, as depicted in Figure 2.4. For the prediction of a 16x16 block, there are 4 possible prediction modes, as depicted in Figure 2.5.

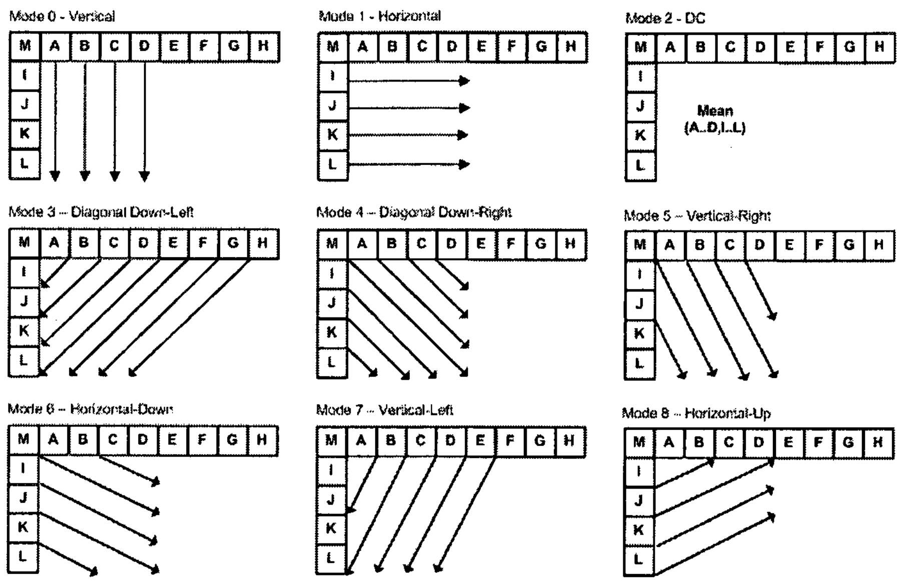


Figure 2.4: Nine possible 4x4 intra prediction modes. Pixels in mode 0 are extrapolated from upper pixels. Pixels in mode 1 are extrapolated from left pixels. Pixels in mode 2 are the average of the upper pixels and left pixels. Pixels in modes 3-8 are a weighted average of the relevant pixels [1].

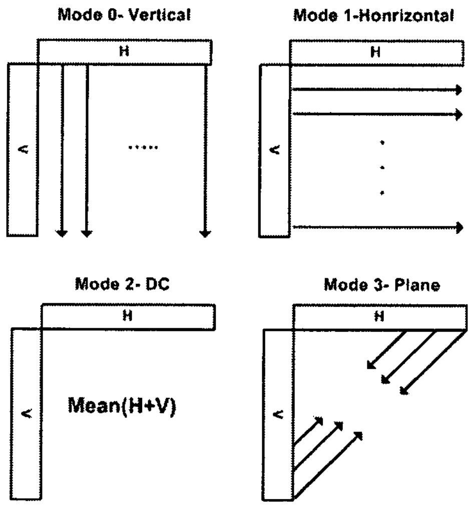


Figure 2.5: Four possible 16x16 intra prediction modes. Pixels in mode 0 are extrapolated from upper pixels. Pixels in mode 1 are extrapolated from left pixels. Pixels in mode 2 are the average of the upper pixels and left pixels. Pixels in mode 3 are predicted from the upper pixels and left pixels using a linear plane function [1].

In summary, each intra prediction mode uses previously encoded pixels above and/or to the left of the current block being predicted. One should note that, bit errors can spatially propagate within a slice when intra prediction is used. One should also note that, in order for a prediction mode to be usable, the pixels used for prediction must be part of the current slice. In the thesis, each slice is encoded as one row of macroblocks. Therefore, all blocks at the top edge of each macroblock can not use upper pixels for prediction since these pixels are not part of the current slice. Similarly, all blocks at the left edge of the first macroblock in the slice can not use left pixels for prediction since these pixels are not part of the current slice. Therefore, there are many blocks in each slice that have unusable prediction modes. Thus, if a bit error causes the H.264 decompressor to apply a prediction mode which is unusable for the current block being predicted, a semantic error is detected. See Section 3.3.1 for details.

2.1.3.4 Entropy Coding

In the H.264/AVC standard, syntax elements within the slice layer and below are entropy coded using either Context Adaptive Variable Length Coding (CAVLC) or Context-based Adaptive Binary Arithmetic Coding (CABAC). Compared to CAVLC, CABAC achieves higher compression efficiency at the price of the increased computational complexity [7].

Following the work in [4], [5], CABAC is chosen for entropy coding in this thesis. The entropy coding process using CABAC consists of three steps: Binarization, Context Modeling and Binary Arithmetic Coding, as depicted in Figure 2.6 [7].

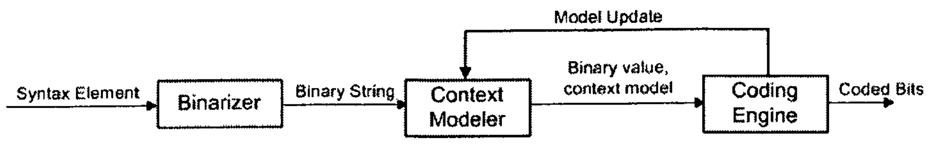


Figure 2.6: Block diagram of CABAC entropy coding process [7].

Binarization is a preprocessing step in which a nonbinary valued H.264 syntax element is mapped onto a unique binary sequence (so-called *bin*).

In context modeling, a probability model is selected for the binarized syntax element. The choice of the probability model depends on the type of the binarized syntax element and in many cases depends on the previously encoded syntax elements.

After the assignment of a context model, the binarized syntax element is encoded using the arithmetic code.

Model update: the selected context model is updated based on the actual coded value (e.g. if the value of a bit in the binarized syntax element is '1', the frequency count of '1' is increased).

One should note when CABAC entropy coding is used, if a bit error occurs in a compressed video bit stream, its encompassing syntax element will be erroneous [4]. And because the decoding of a syntax element is in general dependent on the decoding of previous syntax elements, bit errors often propagate in the H.264 video bit stream when CABAC entropy coding is used, as mentioned in Section 1.1.

2.1.4 H.264 Compressor

As mentioned earlier, the H.264/AVC standard does not define an inflexible video encoding-decoding process [1]. Rather, it defines syntax and semantics elements of the decoded bit stream and their orders in the bit stream, which is well explained in [8]. The H.264 compressor as well as the H.264 decompressor used in this thesis is the

H.264/AVC reference software version JM 9.6 [9]. The block diagram of the H.264 compressor is shown in Figure 2.7.

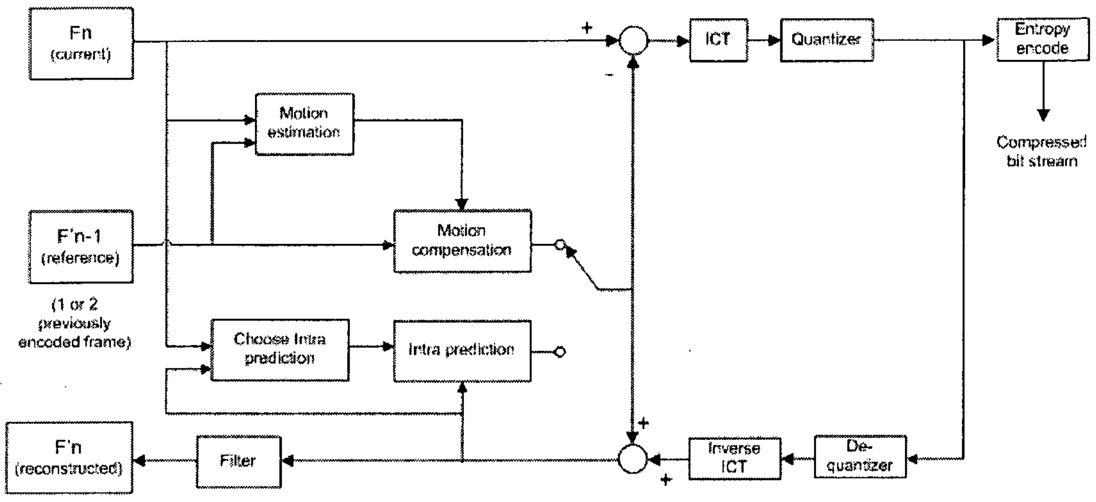


Figure 2.7: H.264 compressor [1].

For an input frame to be encoded, depending on the coded frame type, each macroblock in the frame is predicted using either intra prediction or inter prediction (as discussed in Section 2.1.3.3). The predicted block is subtracted from the original block to produce a residual block. The residual block is then transformed using ICT and quantized. The quantized values are then entropy coded using either CABAC or CAVLC. The entropy-encoded coefficients, together with side information required to decode each macroblock (prediction modes, quantizer parameters, motion vector information, etc) form the compressed video bit stream [1].

In addition to encoding each macroblock, the encoder also decodes it to provide a reference for further predictions. The coefficients are de-quantized and ICT inverse transformed to produce a reconstructed residue. The predicted block is added to the reconstructed residue to create the reconstructed macroblock (a decoded version of the original macroblock). A filter is used to reduce the effects of blocking distortion. After this process has been applied to all macroblocks in a frame, the decoded frame is stored in either List 0 or List 1 for further prediction.

One should note that although compression removes spatial and temporal redundancy in a video sequence, compressed video signals still contain residual redundancy, and this redundancy information can be exploited to correct transmission errors without any extra information being added into a bit stream (See Chapter 3 for further discussion). The more redundancy information that a compressed video sequence has after the compression, the less the compression efficiency is, however, the more resilient the compressed video sequence becomes to transmission errors.

2.1.5 H.264 Decompressor

The block diagram of the H.264 Decompressor is shown in Figure 2.8.

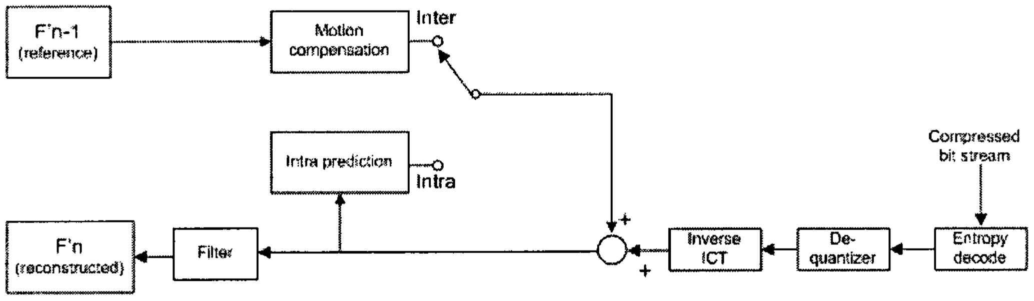


Figure 2.8: H.264 decompressor [1].

The H.264 decompressor first entropy decodes the data for each macroblock to obtain both the prediction information and a set of quantized coefficents. The coefficients are de-quantized and ICT inverse transformed to produce a decompressed residue. The prediction information is added to the decompressed residue. The resulting blocks are then filtered to create decompressed macroblocks. The decompressed macroblocks are stored in either List 0 or List 1 for further prediction.

As will be discussed in Chapter 3, when bit errors occur in an H.264 compressed video bit stream, they usually cause semantic errors and the H.264 decoder is able to detect them. There are several types of semantic errors. Among the most common of these error types are: Intra prediction mode errors, slice fragment errors, slice run-on errors, macroblock run-on errors and illegal reference index errors. See Section 3.3.1 for details.

2.2 Literature Review on Joint Source Channel Decoding for Video Transmission

In order to enhance error resilience of compressed images and videos when transmitted over noisy channels, several authors have recently proposed schemes which exploit the residual redundancy in the compressed data along with the redundancy systematically introduced by the channel coder.

Joint Source Channel Decoding (JSCD) schemes using the Arithmetic Codes (AC) were proposed in [10]-[12]. The input sequence is source encoded using the AC. During the source encoding process, a small interval for a forbidden symbol (which does not belong to the source alphabet) is embedded into the bit stream. Basing on this added redundancy, the AC decoder is able to detect transmission errors. Specifically, if the AC decoder detects the forbidden symbol, it means that transmission errors occur.

Tingjun *et.al.* [10] proposed a JSCD scheme that uses Low-Density Parity-Check (LDPC) codes and AC for the transmission of image/video over an AWGN channel. After AC encoding, the source encoded bit stream is interleaved and channel encoded using LDPC. The coded bit stream is Binary Phase-Shift Keying (BPSK) modulated and transmitted across the AWGN channel. At the receiver side, the LDPC decoder sends the decoding results to the AC decoder. Basing on the added redundancy information, the AC decoder is able to detect errors. The information about the error location provided by the AC decoder is then sent back to the LDPC decoder to correct more transmission errors.

The JSCD scheme proposed in [11] is a concatenated scheme, in which the source decoder and the channel decoder are combined. In particular, the error detection provided by the AC is used as the outer code. A sequential decoder is used as the inner code. This inner code uses the information obtained at the output of the channel along with the error detection capability provided by the AC (the outer code) to perform error correction.

Grangetto *et.al.* [12] proposed a JSCD scheme in which the input sequence is source encoded using the AC. At the receiver side, a MAP (maximum *a-posteriori*) decoder is proposed to decode the information, which unifies the AC decoding and error correction tasks into a single process. The coding redundancy associated with the forbidden symbol is used by the proposed MAP decoder to select the most probable decoded sequence.

The JSCD approach represented in [10]-[12] requires extra bits to be transmitted at the source encoder. Several authors proposed another JSCD approach that does not require bandwidth expansion at the source encoder [13]-[17].

Wang and Yu [13] proposed a joint source channel MAP decoding scheme which is applied to the decoding of motion vectors in an H.264 coded video bit stream. The H.264 motion vectors are modeled as a 1-D Markov processes. This 1-D Markov source is Variable Length Code (VLC) encoded, convolutionally encoded and transmitted over a noisy channel. Note that all syntax elements other than motion vectors are assumed to be error-free. At the receiver side, the MAP decoder uses information from both the channel and the source statistics to select the most probable set of motion vectors.

In Syntax Based Error Concealment (SBEC) [14], the MPEG-2 bit stream is simply channel encoded by adding 1 parity check bit after each 12-information-bit block. At the receiver side, by parity checking, the 13-bit blocks that contain bit errors are located, and are Error Detection (ED) blocks. With the assumption that each ED block has only one error, there are 13 possible correct interpretations. Therefore, the slice containing an ED block is decoded up to 13 times and each time a slice candidate is generated by toggling one different bit. Note that if a slice contains N ED blocks, the number of candidates needs to be generated and syntactically verified is 13^N . These candidates were decompressed and the first one without syntax violation is accepted as the correct bit stream.

Syntax and Discontinuity Based Error Concealment (SDBEC) [15] is an extension of SBEC developed in [14]. SDBEC is the same as SBEC in the sense that it makes use of the syntactic residual redundancy in the video. The difference is, rather than stopping when a syntactically correct slice candidate is found, this scheme keeps decompressing all remaining candidates. After the decompression is complete, a set of seemingly correct candidates for each slice is obtained. A discontinuity measure is proposed to evaluate the spatial smoothness of each candidate in the set. Note that this measure works in the decompressed domain. The smoothest slice candidate is finally chosen as the correct bit stream.

Joint Forward Error Correction and Error Concealment for Compressed Video [16] works with the MPEG-2 videos. The channel code used in the scheme is a block-based (16, 8) quasi-cyclic code. Using the Hamming distance as a measure of slice candidate's likelihood, slice candidates are generated and grouped into three groups: the shortest

Hamming distance group- the one that has the highest priority, the shortest Hamming distance plus 1 group and the shortest Hamming distance plus 2 group- the one that has the lowest priority. These slice candidates are decompressed and verified for syntactic/semantic error. A discontinuity measure is then applied to evaluate the spatial smoothness of the slice candidates that have no syntactic/semantic violation. The best slice candidate is chosen by taking into account two measures: the discontinuity measure and the Hamming distance of each slice candidate. The priority level of each measure is decided experimentally.

The schemes proposed in [13]-[17] do not feed information from the source decoder back to the channel decoder. Schemes proposed in [5], [18]-[20] considers feeding information back to the channel decoder in an iterative manner.

Peng *et.al.* [18] proposed an Iterative Joint Source Channel Decoding (IJSCD) scheme applied for the transmission of vector quantized images, JPEG images and MPEG-1 video data. A turbo code is used as the channel code. The source decoder performs error detection using soft values of bits generated by the channel decoder. Basing on the error detection results, the extrinsic information of bits introduced by one of the two MAP decoders is modified and passed to the other MAP decoder to begin the next iteration.

Pu *et.al.* [19] proposed an IJSCD framework for JPEG2000 transmission. In this scheme, JPEG2000 is used as the source coder and an LDPC code is used as the channel coder. The decoding process is performed iteratively. Specifically, on each iteration, the LDPC decoding is performed. The decoded bit stream is then source decoded by a JPEG2000 decoder. The JPEG2000 decoder is able to detect errors in the bit stream by using built-in error resilience tools. The error detection information from the source decoder is passed to the channel decoder and used to modify the soft values of bits to begin the next iteration.

Another IJSCD schemes were proposed in [20], [5]. The compressor in [20] is MPEG-4 using VLC for entropy coding. A turbo code with two convolutional codes is used as the channel code. Meanwhile, H.264/AVC is used for the compression in [5] with CABAC as the entropy code. A single convolutional code is used as the channel code. For both [20], [5], in each iteration, for each slice, soft values obtained from the output of the channel decoder are used to generate and rank a list of slice candidates in descending order of likelihood. At the source decoder, these slice candidates are verified for semantic correctness and the Best Slice Candidate (BSC) is chosen among them. For both [20], [5], the first slice candidate which passes the verification is chosen as the BSC. In case there's no slice candidate passing the verification, [20] chooses the one with the highest likelihood as the BSC, while [5] chooses the one with the latest failure location as the BSC. In both [20] and [5], soft values of decoded information bits are modified according to hard values of these bits in the BSC and fed back to the channel decoder for the next iteration. On the last iteration, the BSCs of all video slices are decompressed to get a final output video.

The schemes proposed in this thesis are built primarily on the work in [5]. In contrast to the work in [5], slice candidates that failed semantic verification are exploited. The performance of the schemes proposed in the thesis is only compared to the performance of the scheme in [5] since they use the same framework (i.e. the same source coder and channel coder). See Chapter 3 and Chapter 4 for details.

2.3 Summary

This chapter discussed background information about the H.264/AVC video compression standard which is relevant to the work of this thesis. In particular, the structure of the H.264 bit stream and the operation of the H.264 compressor and decompressor were presented. Also, a literature review on the previous work about joint-source channel decoding for the transmission of videos and images was presented.

Chapter 3
Iterative Joint Source-Channel Decoding Using Virtual Checking Method (IJSCD-VC)

Chapter 2 discussed the important coding features of the H.264/AVC standard as well as the mechanism of the H.264 compressor and decompressor. This chapter proposes an IJSCD scheme that makes use of the H.264 source semantics residual redundancy together with soft values of bits produced by the channel decoder to correct transmission errors in a noisy video sequence. A method, called Virtual Checking, is proposed, which uses information of slice candidates that failed semantic verification to virtually check the current slice candidate.

First, conventional error concealment methods are discussed in Section 3.1. The proposed scheme, namely Iterative Joint Source-Channel Decoding using Virtual Checking method (IJSCD-VC) [21] is then discussed in Section 3.2. The operation of the Source Semantic Verifier is described in Section 3.3. Section 3.4 discusses the error detection capability of the Source Semantic Verifier. Next, the Virtual Checking method is presented in Section 3.5. The Modifier is discussed in Section 3.6. The performance of the proposed scheme is assessed objectively and subjectively in Section 3.8. The complexity of the proposed scheme is also evaluated in Section 3.8. Finally, the chapter concludes with a summary in Section 3.9.

3.1 Conventional Error Concealment

As mentioned in Chapter 1, a compressed video bit stream is vulnerable to transmission errors. Errors that occur in one frame can propagate within that frame and to subsequent frames and severely degrade the output visual quality of the decompressed video. One mechanism that causes the error propagation effect is the use of entropy coding since a single bit error can result not just in an error in the symbol currently being decoded but also in subsequent symbols due to the loss in synchronization [3], [13]. Another cause of error propagation is the use of spatial-temporal prediction in video coding. Specifically, bit errors in an erroneously decoded portion of an I-frame or P-frame can spatially/temporally propagate if it is used as a reference sample for the currently decoded frame.

To cope with the vulnerability of compressed videos when transmitted over noisy channels, many error concealment techniques have been proposed [22]-[37]. In general, the two basic types of error concealment techniques are: spatial error concealment and temporal error concealment. These two techniques involve estimating the lost pixels due to transmission errors by exploiting the temporal/spatial correlation in a compressed video bit stream and do not require any additional bits to be transmitted [3].

In this thesis, when the proposed schemes fail to correct errors, temporal replacement - one of the simple temporal error concealment methods is used. Basically, in case an error is detected during the final decompression, this method replaces each macroblock in the current slice from the point where the error is detected to the end of slice by a corresponding macroblock at the same spatial location from the previously decoded frame [4]. This simple concealment does not take into account motion compensation and hence shifts in the concealed picture will be visible if there is motion. Note that temporal replacement is also used as a benchmark for every proposed scheme in this thesis.

3.2 Iterative Joint Source-Channel Decoding Using Virtual Checking Method (IJSCD-VC)

In this chapter, the IJSCD-VC scheme is proposed to enhance the error robustness of H.264 compressed video sequences when transmitted over a noisy channel. This scheme follows the previous work [5], [20] in exploiting both the redundancy systematically added by the channel coding and the semantic residual redundancy in the compressed video, in an iterative manner, to correct transmission errors. The block diagram of the IJSCD-VC scheme is shown in Figure 3.1.

The pseudocode explaining how the scheme works is as follows.

At the transmitter side:

```
1. Compress the raw video sequence using CABAC entropy coding
2. Split the compressed video sequence to two parts: header stream (assumed to be
   error-free) and data stream
3. Interleave the data stream
4. Encode the data stream by a convolutional code
5. Modulate the encoded bit stream using BPSK
6. Transmit the modulated bit stream over an AWGN channel
```

At the receiver side:

```
1. FOR i=1: num_iteration // num_iteration is the total number of iterations
   1.1.Decode the received bit stream by a MAP decoder
   1.2.Deinterleave the decoded bit stream
```

```
    1.3. Merge the header stream with the deinterleaved bit stream
    1.4. Generate a list of slice candidates for each slice by using soft values of bits
    1.5. Verify the semantic validity of these slice candidates to choose the best slice
         candidate for each slice
    1.6. Modify soft values of bits in each slice by using the information extracted from
         the semantic verification process
    1.7. Interleave the modified soft valued bit stream
    1.8. Feed the interleaved bit stream back to the channel decoder for the next
         decoding iteration
    ENDFOR
2. Send the best slice candidate of each slice to the H.264 decompressor
3. Decompress to get the output video sequence
```

The remainder of this section describes the modules in the IJSCD-VC scheme with the exception of the H.264 compressor and H.264 decompressor, which were discussed in Section 2.1.4 and 2.1.5 respectively.

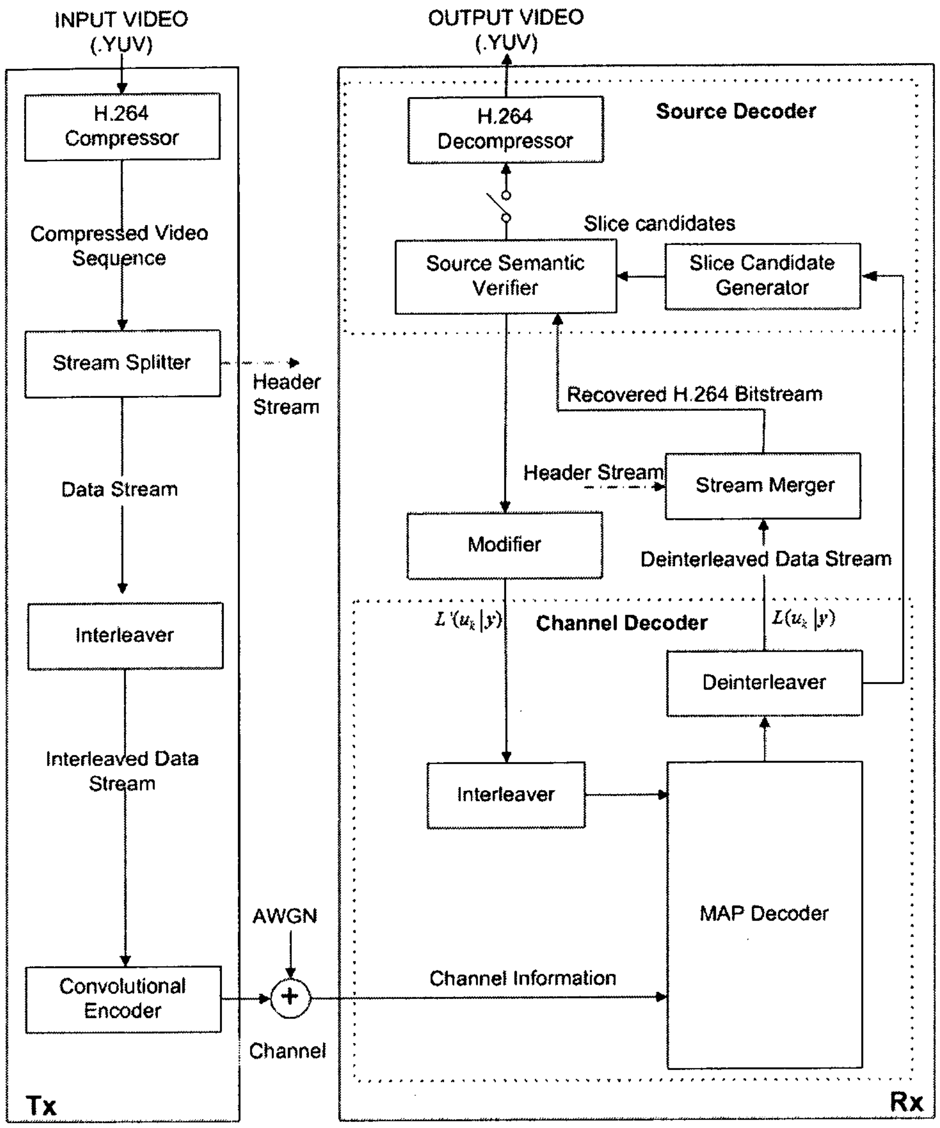


Figure 3.1: Block diagram of the IJSCD-VC scheme.

3.2.1 Stream Splitter and Stream Merger

As discussed in Chapter 2, an H.264 bit stream is hierarchically organized into different layers: sequence, GOP, frame, slice, macroblock and block. The task of the Stream Splitter is to split the compressed video bit stream into two parts: the header stream which includes all bits in the slice headers and higher layers; and the data stream which includes the remaining slice data bits in the video sequence [5]. The header stream is assumed to be protected by a very strong code and consequently error-free. By doing this the header information in the compressed video is not damaged or lost and so the decompression process can be performed properly. The data stream, on the other hand, is sent over a noisy channel.

At the receiver side, the Stream Merger combines the noisy data stream with the header stream to produce a noisy H.264 video sequence. This video sequence can now be displayed using software that supports H.264 standard such as VLC media player.

3.2.2 Interleaver and Deinterleaver

In convolutional decoding, incorrect decoding decisions often make errors appear as "bursts", that is when an information block is in error, several bits in it are erroneous. Consequently, it is often the case that there are many errors in some slices while others are error-free. In such situations, the IJSCD-VC scheme does not work effectively. Specifically, for slices that are severely damaged by errors, it is difficult for the IJSCD-VC scheme to correct them properly. Moreover, for slices that are error-free, applying IJSCD-VC scheme can not yield better result either (when compared with convolutional decoding only). Hence, it is important that errors occurring in a particular convolutional decoding block should be spread out among many slices so that each slice only suffers from a manageable number of errors.

To effect this, an Interleaver is used. At the transmitter side it reorders the bits in a predetermined manner.

At the receiver side, the Deinterleaver reconstructs slices by collecting bits from multiple convolutional decoding blocks and restoring their original order.

Generally, common interleaving techniques include the block interleaver, the random interleaver and the convolutional interleaver. In this thesis, a block interleaver with a 5000-by-10 matrix is used.

3.2.3 Convolutional Encoder

A constraint length of 2, rate ½ Recursive Systematic Convolutional code with generator (1, 5/7) is used in this thesis. The generator matrix G(D) is chosen as follows [38]:

$$\mathbf{G}(D) = \begin{bmatrix} 1 & \dfrac{1+D^2}{1+D+D^2} \end{bmatrix} \tag{3.1}$$

The corresponding controller canonical form is shown in Figure 3.2.

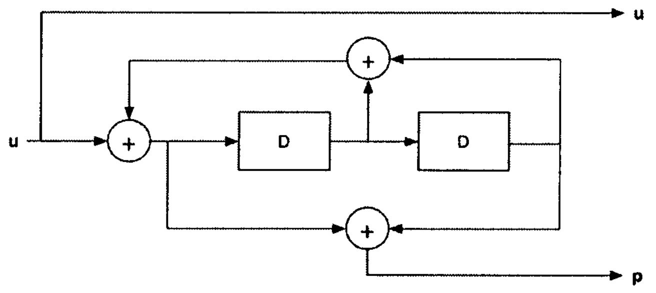


Figure 3.2: Controller canonical form of the Convolutional Encoder [4], [38].

The data stream after being interleaved is divided into information blocks of 10000 bits, which are then encoded one after another. For each information block to be encoded, four 0's are padded at the end of it in order to reset the encoder state to zero state after the encoding process is completed. By doing this, it is certain that the subsequent information blocks will be encoded with a refreshed encoder state. Therefore the actual length of each information block is 10004 bits and after being encoded, each coded block consists of 10000 information bits, 10000 parity check bits and 8 padded bits. Due to the use of a systematic convolutional code, the structure of the coded bit stream is: one information bit followed by one parity check bit and so on. The coded bit stream is then BPSK modulated and sent over a noisy channel.

3.2.4 Channel Model

In this thesis, an AWGN channel with noise variance σ^2 is assumed.

3.2.5 MAP Decoder

The decoding technique used at the channel decoder is the MAP algorithm which utilizes soft-input soft-output values [39]. The MAP decoding algorithm for rate ½ Recursive Systematic Convolutional (RSC) code with AWGN channel and BPSK modulation described as follows comes from [39]. Denote:

- m: the RSC encoder's memory
- S: the set of all 2^m states of the RSC encoder
- s_k: the state of the encoder at time k
- $\boldsymbol{u} = (u_1, u_2, ..., u_k, ..., u_N) \in [0,1]^N$: the information bit sequence
- $\boldsymbol{p} = (p_1, p_2, ..., p_k, ..., p_N) \in [0,1]^N$: the parity check bit sequence
- $\boldsymbol{x}^s = (x_1^s, x_2^s, ..., x_N^s) \in [-1,1]^N$: the BPSK modulated information bit sequence
- $\boldsymbol{x}^p = (x_1^p, x_2^p, ..., x_N^p) \in [-1,1]^N$: the BPSK modulated parity check bit sequence
- $\boldsymbol{x} = (x_1^s, x_1^p, x_2^s, x_2^p ..., x_N^s, x_N^p) \in [-1,1]^{2N}$: the modulated coded bit sequence. It includes $\boldsymbol{x}^s$ and $\boldsymbol{x}^p$.
- $\boldsymbol{y}^s = (y_1^s, y_2^s, ..., y_N^s) \in \Re^N$: the noisy version of $\boldsymbol{x}^s$
- $\boldsymbol{y}^p = (y_1^p, y_2^p, ..., y_N^p) \in \Re^N$: the noisy version of $\boldsymbol{x}^p$

- $y = (y_1^s, y_1^p, y_2^s, y_2^p, \ldots, y_N^s, y_N^p) \in \Re^{2N}$: the received noisy bit sequence after the channel. It includes y^s and y^p.

The MAP decoder examines the received sequence and computes the a *posteriori* probability (APP) of bit u_k (where k is the time index), which is defined as:

$$L(u_k|y) = \ln\left(\frac{P(u_k = 1|y)}{P(u_k = 0|y)}\right) \tag{3.2}$$

Here, $P(u_k = 1|y)$ is the probability that the decoded bit $u_k = 1$ given that y is received. $P(u_k = 0|y)$ is similarly defined.

Incorporating the trellis of the RSC encoder, with some manipulations, Equation 3.2 can be rewritten as:

$$L(u_k|y) = \ln\left(\frac{\sum_{(s',s)\in S^{(1)}} \alpha_{k-1}(s')\gamma_k(s',s)\beta_k(s)}{\sum_{(s',s)\in S^{(0)}} \alpha_{k-1}(s')\gamma_k(s',s)\beta_k(s)}\right) \tag{3.3}$$

Here, $S^{(1)}$ is the set of the pairs of states (s', s) such that the transition from $S_{k-1} = s'$ to $S_k = s$ is caused by the input $u_k = 1$. $S^{(0)}$ is similarly defined for $u_k = 0$.

In Equation 3.3, $\alpha_k(s)$ and $\beta_k(s)$ are called the *forward metric* and the *backward metric* respectively. These two metrics are recursively calculated as:

$$\alpha_k(s) = \sum_{\text{all } s'} \alpha_{k-1}(s')\gamma_k(s',s) \tag{3.4}$$

$$\beta_k(s) = \sum_{\text{all } s''} \beta_{k+1}(s'')\gamma_{k+1}(s,s'') \tag{3.5}$$

Here, $\gamma_k(s',s)$ is called the *branch metric,* defined as

$$\gamma_k(s',s) = P(S_k = s; y_k | S_{k-1} = s') \tag{3.6}$$

After some manipulation, $\gamma_k(s',s)$ is calculated as:

$$\gamma_k(s',s) = A_k B_k \exp\left[\frac{1}{2} x_k^s \left(L(u_k) + L_C y_k^s\right)\right] \gamma_k^e(s',s) \tag{3.7}$$

Where:

$$A_k = \left(\frac{1}{2\pi\sigma^2}\right) \exp\left[-\frac{(y_k^s)^2 + (y_k^p)^2 + 2}{2\sigma^2}\right] \tag{3.8}$$

$$B_k = \left(\frac{e^{-L(u_k)/2}}{1 + e^{-L(u_k)/2}}\right) \tag{3.9}$$

$$\gamma_k^e(s',s) = \exp\left[\frac{1}{2} L_C x_k^p y_k^p\right] \tag{3.10}$$

Here, $L(u_k)$ is called the *a priori* information which is defined as follows:

$$L(u_k) = \ln\left(\frac{P(u_k = 1)}{P(u_k = 0)}\right) \tag{3.11}$$

And L_C is called the channel reliability

$$L_C = \frac{2}{\sigma^2} \tag{3.12}$$

After incorporating the *a priori* information, the soft value of bit u_k can be further represented as:

$$L(u_k | y) = \underbrace{L(u_k)}_{\substack{a\ priori \\ \text{information}}} + \underbrace{L_C . y_k^s}_{\text{channel value}} + \underbrace{\ln\left(\frac{\sum_{(s',s)\in S^{(1)}} \alpha_{k-1}(s')\gamma_k^e(s',s)\beta_k(s)}{\sum_{(s',s)\in S^{(0)}} \alpha_{k-1}(s')\gamma_k^e(s',s)\beta_k(s)}\right)}_{\text{extrinsic information}} \tag{3.13}$$

In summary, the soft value of bit u_k obtained at the output of the MAP decoder comprises of three terms: the *a priori* information of bit u_k, the channel value and the extrinsic information of bit u_k (denoted $L_e(u_k)$).

On the other hand, given that $L(u_k|y)$ is obtained at the output of the MAP decoder (Equation 3.13), and that $P(u_k=1|y)+P(u_k=0|y)=1$, the *a posteriori* probability of $u_k=1$ and $u_k=0$ can be calculated as follows:

$$\begin{aligned} P(u_k=1|y) &= \frac{e^{L(u_k|y)}}{1+e^{L(u_k|y)}} \\ P(u_k=0|y) &= \frac{1}{1+e^{L(u_k|y)}} \end{aligned} \tag{3.14}$$

Basing on Equation 3.14, the MAP decoder decides $u_k=1$ if $P(u_k=1|y) \geq P(u_k=0|y)$ and $u_k=0$ if $P(u_k=1|y) < P(u_k=0|y)$. In other words, the sign of the soft value $L(u_k|y)$ will indicate whether u_k is 1 or 0. Furthermore, the magnitude of $L(u_k|y)$ implies how certain the hard decision made on bit u_k would be. If $L(u_k|y) \approx 0$, this leads to the fact that $P(u_k=1|y) \approx P(u_k=0|y)$. Hence the decision made on u_k is not certain. Vice versa, if $L(u_k|y) >> 0$ or $L(u_k|y) << 0$, this corresponds to the fact that $P(u_k=1|y) >> P(u_k=0|y)$ or $P(u_k=1|y) << P(u_k=0|y)$. Consequently it is almost certain about the decision made on u_k. It is obvious that the higher the magnitude of $L(u_k|y)$, the more certain the decision made on u_k.

The soft values of all decoded information bits are used in the Slice Candidate Generator module in order to generate slice candidates, which is to be described in Section 3.2.6.

3.2.6 Slice Candidate Generator (SCG)

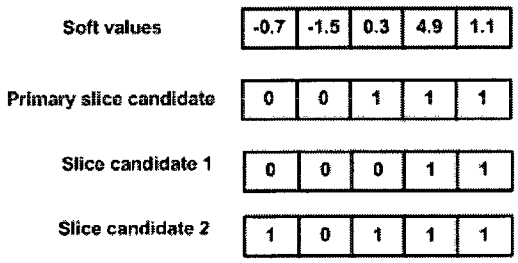
3.2.6.1 The General Idea

As mentioned in Chapter 2, each slice in the H.264 compressed video sequence is decoded independently [6]. The advantage of this decoding mechanism is that the effect of error propagation between slices in the same frame caused by the loss of synchronization can be eliminated. In light of this, the proposed error correction schemes in this thesis are done on a slice by slice basis, i.e. not on a macroblock basis or frame basis. Note that a raw video sequence in this thesis is compressed in such a way that each slice contains one row of macroblocks.

When transmitted over a noisy channel, the compressed bit stream is contaminated by noise and hence errors occur. For each slice, the SCG tries to recover the original error-free slice (which is called the Target Candidate) from its noisy version by flipping one or more bits which are suspected of being erroneous. As discussed in Section 3.2.5, the soft value of each bit produced by the MAP decoder indicates the most likely value for the bit in question as well as its reliability. Hence, bits that have small absolute soft values are more likely to be erroneous than bits that have large absolute soft values.

When the hard decisions are done for all decoded information bits in the slice, the most likely slice candidate is generated. This is called the Primary Slice Candidate (PSC). A slice candidate is defined by a set of bits that are different or "flipped" from the PSC.

These are called the flip-bits. By definition the PSC has no flip bits. An example illustrating the concept of a slice candidate is shown in Figure 3.3. In this example, Slice candidate 1 has one flip-bit at position 3 in the bit stream. Slice candidate 2 has one flip-bit at position 1 in the bit stream.



Soft values	-0.7	-1.5	0.3	4.9	1.1
Primary slice candidate	0	0	1	1	1
Slice candidate 1	0	0	0	1	1
Slice candidate 2	1	0	1	1	1

Figure 3.3: An example of slice candidates.

The main task of the SCG is to generate a list of slice candidates for each slice with the expectation that the Target Candidate is listed as high as possible on the list [4]. If a slice is N bits long, one way to make sure the Target Candidate is included in the list is to generate all 2^N slice candidates. However, this method's complexity is too high. For example, with a 300-bit-long slice, the total number of slice candidates must be generated for that slice is more than $2*10^{90}$.

Previous work [4], [40], proposed a method of choosing a certain number of the most likely slice candidates by using the soft values of bits in the slice. These are ranked in descending order of likelihood based on the soft values of flip-bits.

Section 3.2.6.2 discusses how to mathematically rank slice candidates in descending order of likelihood as what was presented in [4], [40]. Section 3.2.6.3 discusses how many slice candidates should be generated for each slice.

3.2.6.2 Ranking Slice Candidates

For a given slice of length N, denote:

- ❖ $s = (s_1, s_2, ..., s_N) \in [0,1]^N$: a particular slice candidate
- ❖ $v = (v_1, v_2, ..., v_N) \in [0,1]^N$: the Primary Slice Candidate (PSC)

Given that the APP of each decoded information bit is available by Equation 3.14, in conjunction with the assumption that bits in the same slice are independent, the likelihood of the PSC, denoted $P(v)$, can be calculated as the product of the APP of its bits:

$$P(v) = \prod_{k=1}^{N} P(v_k = u_k \mid y) \qquad (3.15)$$

Where $P(v_k = u_k \mid y)$ is the APP of bit u_k. Note that $P(v_k = u_k \mid y) > P(v_k = \bar{u}_k \mid y)$, in which $\bar{u}_k$ is the inversed value of u_k (i.e. if $u_k = 0$ then $\bar{u}_k = 1$ and vice versa). In other words, if the hard decision made for u_k is 1 then $P(v_k = 1 \mid y) > P(v_k = 0 \mid y)$. Otherwise, if the hard decision is 0 then $P(v_k = 1 \mid y) < P(v_k = 0 \mid y)$. Obviously, the likelihood of the PSC is highest among slice candidates.

Similarly, the likelihood of slice candidate s, denoted $P(s)$, is defined as follows:

$$P(s) = \prod_{k=1}^{N} P(s_k = u_k \mid y) \qquad (3.16)$$

The bit stream in slice candidate s only differs from the one in the PSC at flip-bits' positions. Denote F_S the set of indices to the flip-bits of s, which can be mathematically represented as follows [4]:

$$k \in F_S \text{ if } s_k \oplus v_k = 1 \qquad (3.17)$$

Where $\oplus$ is the exclusive-or operation.

A bit k in slice candidate s can be represented as:

$$s_k = \begin{cases} v_k, & k \notin F_s \\ \bar{v}_k, & k \in F_s \end{cases} \tag{3.18}$$

Here, $\bar{v}_k$ is the inversed value of v_k .The bit stream of slice candidate s can be conceptually divided into two parts: one part is identical to the PSC, and the other one is the flipped version of the PSC. Therefore, Equation 3.16 can be rewritten as follows:

$$P(s) = \prod_{k \notin F_S} P(s_k = u_k \mid y) \prod_{k \in F_S} P(s_k = u_k \mid y) \tag{3.19}$$

Combining with Equation 3.18, Equation 3.19 can be rewritten as:

$$P(s) = \prod_{k \notin F_S} P(v_k = u_k \mid y) \prod_{k \in F_S} P(v_k = \bar{u}_k \mid y) \tag{3.20}$$

Denote $R(\mathbf{s})$ the rank of slice candidate s. It is calculated by:

$$\begin{aligned} R(\mathbf{s}) = -\ln\left(\frac{P(s)}{P(v)}\right) &= -\ln\left(\frac{\prod_{k \notin F_S} P(v_k = u_k \mid y) \prod_{k \in F_S} P(v_k = \bar{u}_k \mid y)}{\prod_{k \notin F_S} P(v_k = u_k \mid y) \prod_{k \in F_S} P(v_k = u_k \mid y)}\right) \\ &= -\ln\left(\prod_{k \in F_S} \frac{P(v_k = \bar{u}_k \mid y)}{P(v_k = u_k \mid y)}\right) \\ &= -\sum_{k \in F_s} \ln\left(\frac{P(v_k = \bar{u}_k \mid y)}{P(v_k = u_k \mid y)}\right) \\ &= \sum_{k \in F_s} \ln\left(\frac{P(v_k = u_k \mid y)}{P(v_k = \bar{u}_k \mid y)}\right) \end{aligned} \tag{3.21}$$

Equation 3.21 shows that, ranking a slice candidate s in descending order of $P(s)$ can now refer to ranking it in ascending order of $R(\mathbf{s})$ due to the relation $P(s) = \frac{P(v)}{e^{R(\mathbf{s})}}$.

Note that $R(\mathbf{s})$ is always a positive number since the condition $P\left(v_k = u_k \mid y\right) > P(v_k = \bar{u}_k \mid y)$ is always satisfied. In order to avoid any confusion, $R(\mathbf{s})$ can be rewritten as follows:

$$R(\mathbf{s}) = \sum_{k \in F_s} \left| \ln\left(\frac{P\left(v_k = u_k \mid y\right)}{P(v_k = \bar{u}_k \mid y)} \right) \right| \tag{3.22}$$

Finally, the rank of the slice candidate s can be written as:

$$R(\mathbf{s}) = \sum_{k \in F_s} \left| L(u_k \,|y) \right| \tag{3.23}$$

In summary, the rank $R(\mathbf{s})$ of slice candidate s is the sum of the absolute soft values of its flip-bits. The smaller the rank is, the more likely the slice candidate is. One should note that the PSC has no flip-bits, hence its rank is zero.

3.2.6.3 The Choice of the Number of Slice Candidates

The Incomplete Partial Sums Algorithm (IPSA) which was well described in [4] is used to generate the N_{SC} most likely slice candidates and put them in descending order of likelihood. Here, N_{SC} is a number chosen by taking into account the required computational complexity. If N_{SC} is chosen to be too small, the probability of having the Target Candidate included in the list is reduced. In contrast, if N_{SC} is larger, the probability of having the Target Candidate included in the list is higher, but the computational complexity is also higher.

The N_{SC} most likely slice candidates generated for each slice are then sent to the next module - the Source Semantic Verifier.

3.3 The Source Semantic Verifier (SSV)

As mentioned in Chapter 2, the H.264/AVC standard does not define an inflexible video encoding-decoding process [1]. Rather, it defines syntax and semantics elements of the decoded bit stream and their orders in the bit stream, which is well explained in [8]. By that, the compressed bit stream encoded by a compliant encoder can be decodable by following the decoding process defined in the standard [1]. Therefore, if an error occurs and violates the syntax/semantics structure of the compressed bit stream, the H.264 decompressor is able to detect it. Section 3.3.1 describes several types of semantic error detected by the SSV. Section 3.3.2 discusses the operation mechanism of the SSV as in [4], [5].

3.3.1 Detecting Errors in H.264

In general, there are two kinds of errors that can be detected by the H.264 decompressor: syntactical errors and semantic errors. If an error causes an invalid entropy decoded syntax element, a syntactical error is detected. On the other hand, if an error causes the H.264 decompressor to execute an incorrect task, this is a semantic error [4].

As mentioned in Chapter 2, the entropy code used in this thesis is CABAC. Since all syntax elements in CABAC are valid, there are no CABAC syntax errors. Therefore, when CABAC entropy coding is used, the only errors that can be detected by the H.264 decompressor are semantic errors [4].

The Detection Bit (DB) is a term used to refer to the bit at which the semantic error is detected. The actual location of the bit error(s) is between the Start of Slice (SOS) and the DB. Note that when the first semantic error is detected in a slice, means this slice is known to be erroneous, the H.264 decompressor will stop checking for more semantic errors in this slice. Therefore, each slice has at most one semantic error [4].

Generally, there are five common types of semantics error that can be detected in H.264, which are well described in [4], [41]: Intra prediction mode errors, slice fragment errors, slice run-on errors, macroblock run-on errors and illegal reference index errors. These types of semantic errors can be classified into three different categories as shown in Figure 3.4 below.

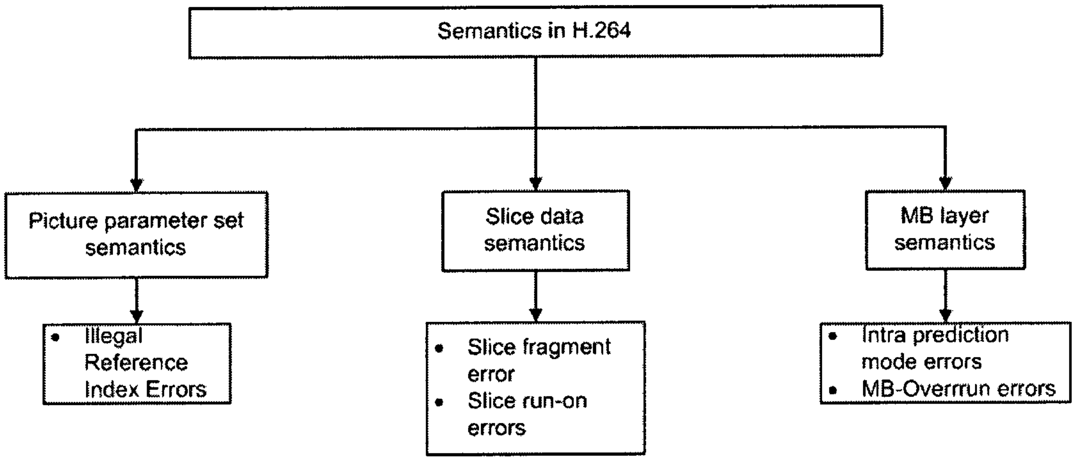


Figure 3.4: Types of semantic errors in H.264/AVC standard.

Intra Prediction Mode Errors

As discussed in Chapter 2, the H.264/AVC standard provides nine intra prediction modes for each 4x4 luma block, four intra prediction modes for each 16x16 luma block, and four intra prediction modes for each chroma component. An intra prediction mode error occurs if the decompressor is instructed to use an unavailable intra prediction mode [4], [41]. For instance, horizontal prediction mode is unusable on a block at the leftmost of a slice since all pixels located at the left side of this block are not from the same slice (see Figure 3.5). So, if a 4x4 luma block at the leftmost of a slice is instructed to use the horizontal prediction mode, an intra prediction mode error is detected.

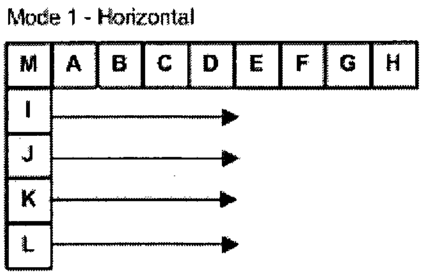


Figure 3.5: Horizontal prediction mode for 4x4 luma block: the left samples I, J, K, L are extrapolated horizontally [1].

Slice Fragment Errors

The *end_of_slice_flag* is a parameter in the H.264 decompressor that signals whether the decompressor reaches the end of the current slice or not [8]. If *end_of_slice_flag* equals to 0, the decompressor is informed that another macroblock is following in the slice. Otherwise, if *end_of_slice_flag* equals to 1, it specifies that the End of Slice (EOS) is reached and that no further macroblock follows. In this case, all remaining bits in the slice, if exist, are ignored by the decompressor.

If the decompressor does not reach the last byte of the slice while the *end_of_slice_flag* equals to 1, a slice fragment error is detected.

Slice Run-On Errors

In contrast to slice fragment errors, a slice run-on error is detected when the decompressor reaches the start code of the next slice while the *end_of_slice_flag* equals to 0. In other words, the decompressor has already reached the end of the current slice and moved on to the next slice while the *end_of_slice_flag* still signals that another macroblock is following in the current slice.

Macroblock-Overrun Errors

This kind of error happens if the number of macroblocks decoded in the current slice is not correct when the decompressor has already reached the EOS.

Illegal Reference Index Errors

In the H.264/AVC standard, previously decoded frames can be used as reference frames and are organized into two lists: *list 0* and *list 1* [1]. While P-macroblocks are predicted using one reference frame stored in *list 0*, B-macroblocks are predicted using one or two reference frame(s) taken from *list 0* and/or *list 1*. These two lists are not always full. Therefore, during the inter prediction process to reconstruct a P-macroblock or B-macroblock, if the decompressor is instructed to use an index to an empty element in *list 0* and/or *list 1*, an illegal reference index error is detected [4], [41].

Simulations in [4], [41] show that, nearly all detected errors in I-slices are intra prediction mode errors, whereas the majority of detected errors in P-slices and B-slices are macroblock-overrun errors.

3.3.2 The Operation of the SSV

The task of the SSV is to verify the semantic correctness of slice candidates in the list, one by one from the most likely (the PSC) to the least likely. Semantic verification involves a partial decoding of the bit stream where the semantic correctness is checked but a fully decompressed video is not generated. The SSV is developed based on the H.264 video codec software provided by [9].

During the semantic verification process, if there is a slice candidate that passes verification, it is chosen as the BSC and the SSV stops verifying the remaining slice candidates in the list. This makes sense since slice candidates are ranked in descending order of likelihood, therefore the first semantically correct slice candidate is the one which is the most similar to the PSC.

In contrast, if the SSV has verified all slice candidates in the list and none of them passes verification, the location where the semantic error is detected for each slice candidate is used as a measure to choose the BSC. In particular, the slice candidate of which DB is detected latest is chosen as the BSC. In other words, the slice candidate that has the DB nearest to the EOS is chosen as the BSC. This is because, if a DB is detected near the SOS, it is more likely that the true bit error locates near the SOS. Hence, it is more likely that this bit error can propagate to the EOS and cause significant visual quality degradation. In contrast, if a DB is detected near the EOS, it is more likely that the true bit error can only propagate from that point to the EOS, and thus the damage caused by the error propagation is less severe.

3.4 Error Detection Capability of the SSV

This section discusses the error detection capability of the SSV. In particular, if an error occurs in a slice, how probable is that it is detected? The question whether or not the location of an erroneous bit influences the probability that it is detected is also discussed in this section.

3.4.1 Estimating the Error Detection Capability of the SSV

In this section, the capability of detecting errors of the SSV is investigated as what has been done in [4], [41]. If a slice has a semantic error, there must be at least one error locating somewhere between the SOS and the DB. However, if one or more errors happen in a slice, it's not always the case that there is a semantic error detected in that slice. In other words, it is admitted that the SSV can not detect all errors occurring in a slice. An experiment is set up to investigate the capability of detecting errors of the SSV.

The experiment is repeated as in [4], [41] by first inserting one error into the Data Stream of the compressed video bit stream for each run. The noisy bit stream is then verified for semantic errors. Observations are made to investigate whether the SSV is able to detect the error or not. The probability of detecting errors of the SSV is calculated as the number of times it detects a semantic error divided by the total number of runs of the experiment.

Simulations are run for two video sequences: Football and Table-Tennis. Each sequence is 4-second in length and has a frame size of 352x240. They are compressed using Main profile, CABAC entropy coding with a Group of Picture (GOP) of length 10 and IPPPP structure.

Table 3.1 shows the simulation results for the two video sequences "Football" and "Table-Tennis". It is observed that for both video sequences "Football" and "Table-Tennis", the SSV can detect almost 99% of bit errors, and less than 1% of errors are undetected.

Table 3.1:The probability of undetected bit errors for two video sequences "Football" and "Table-Tennis" when each simulation run contains one bit error

Video	# Runs	Number of Undetected Errors	Probability of Undetected Errors
Football	1108947	7097	0.0064
Table-Tennis	1231188	8125	0.0066

The experiment results imply that if the H.264 decompressor does not detect any semantic errors in a slice, this slice has a high probability (over 99%) of not having any bit errors [41].

3.4.2 The Relation between the Bit Error's Location and the Error Detection Capability of the SSV

This section further investigates the relation between the locations of bit errors in a slice with the detection capability of the SSV. It is hypothesized that the locations of bits in a slice influence the error detection capability of the SSV for this slice. Specifically, it is believed that the missed error detection is more likely to happen if bit errors arise near the EOS. In other words, it is believed that errors occurring close to the SOS are more likely to be detected than errors occurring close to the EOS.

An experiment is repeated as in [4], [41] to verify this hypothesis. The two compressed video sequences "Football" and "Table-Tennis" are used as the inputs for this experiment. For each run, a bit error is inserted into each slice. If this bit error is not detected by the SSV, the distance in bits between the bit error and the EOS is recorded. At the end of the experiment, histograms showing the probability of undetected errors versus the distance (in bits) from them to the EOS are plotted.

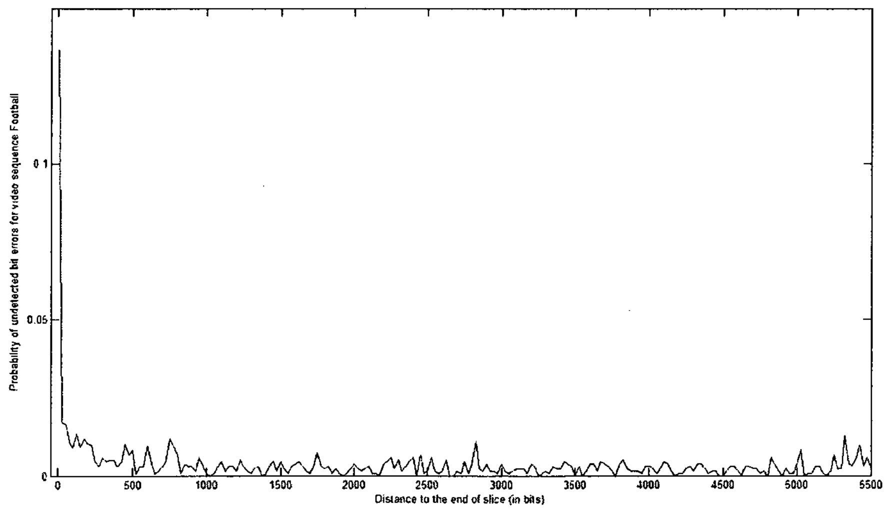


Figure 3.6: The probability of undetected errors as a function to the EOS for video sequence "Football".

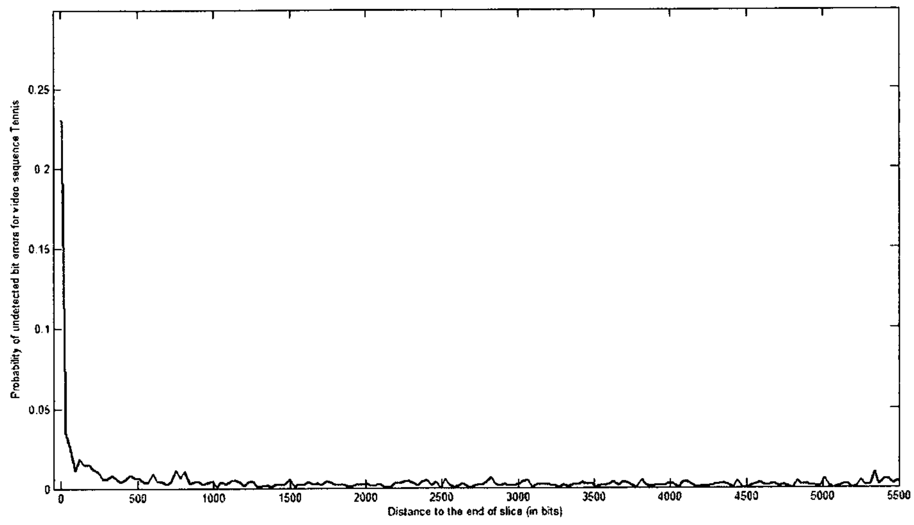


Figure 3.7: The probability of undetected errors as a function to the EOS for video sequence "Table-Tennis".

Figure 3.6 and Figure 3.7 show the probability of undetected bit errors as a function of the distance between those bit errors and the EOS for the two video sequences "Football" and "Table-Tennis" respectively. The results show that, for both "Football" and "Table-Tennis", the probability of not detecting an error increases precipitously near the EOS. One of the reasons to explain this phenomenon is, in H.264/AVC standard, some 0's bits are padded at the end of each slice in order to make sure that slices are byte-aligned (as discussed in Chapter 2). These redundant bits have no effect on either the decompression process or the output visual quality and in case they are damaged by noise, they cannot be detected by the SSV.

Observations made in [4], [41] showed that undetected bit errors do not cause any degradation in terms of visual quality and PSNR measurement. This means that, in terms of visual assessment, it is acceptable to display a video slice which is semantically correct although it still contains undetected bit errors.

3.5 Virtual Checking Method (VC)

As discussed in Section 3.3.2, the task of the SSV is to find the BSC for each slice. The SSV in previous work [5], [20] verifies slice candidates in the list, one by one from the most likely to the least likely until it finds the one that does not cause any semantic error or until it verifies all slice candidates in the list.

In this section, a new checking method, namely Virtual Checking (VC) is proposed to accelerate the work of the SSV. Using VC, slice candidates are not sequentially verified as in [5], [20]. In fact, it is possible to eliminate some slice candidates without verifying their semantics by relying on the results of previous slice candidates that failed semantic verification. This can be illustrated with an example depicted by Figure 3.8.

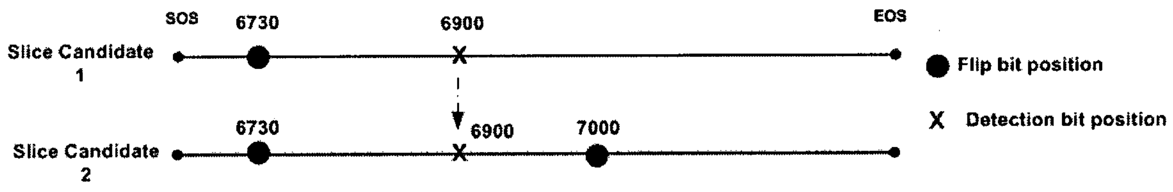


Figure 3.8: An example of Virtual Checking method.

Two slice candidates (which are uniquely defined by their sets of flip-bits F_S) are to be verified for semantic errors. Slice Candidate 1 flips the bit at position 6730. After attempting verification, a DB is detected at 6900. Slice Candidate 2 flips two bits at 6730 and 7000. One can see that the two candidates are identical from the SOS to the DB. Hence, it is not necessary to attempt to verify the semantic correctness of Slice Candidate 2. This is because it is known *a-priori* that Slice Candidate 2 will fail the semantic verification in exactly the same way as Slice Candidate 1, i.e. the DB of Slice Candidate 2 would be at 6900.

In summary, VC uses semantic verification results of previous slice candidates that failed verification to investigate the semantic correctness of the current slice candidate without running the verification. Specifically, whenever a slice candidate to be verified is identical to a previously verified (and failed) slice candidate, up to the DB of that previously verified slice candidate then it is known that the current slice candidate will fail the semantic verification in exactly the same way. This slice candidate is said to be virtually checked.

VC can be used to speed up the work of the SSV while yielding the same performance. In other words, when VC is used, the number of slice candidates actually verified is in general reduced while the BSC chosen for each slice is the same as the case when VC is not used. This is called the Fast Search (IJSCD-VC-FS) scheme.

Alternatively, VC can achieve better performance while keeping almost the same complexity. This is because with the same number of slice candidates actually tested by the SSV (300 slice candidates, for example), the number virtually checked can be much larger (one should also note that checking semantics for a slice candidate takes much more time than generating it). Thus, the probability of finding the BSCs which are semantically correct is higher. This is called the Performance Improvement (IJSCD-VC-PI) scheme. Denote N_{MAX} the maximum number of slice candidates generated for each slice when VC is used. Then the SSV using VC can virtually check up to N_{MAX} slice candidates to find the BSC for each slice. For the simulation, N_{MAX} is empirically chosen.

3.6 Modifier

After the SSV is completed, information about the BSC as well as the soft value of each bit is passed to the Modifier. The IJSCD-VC scheme follows the Modifier proposed in [5] to modify the soft value of bit u_k (where k is the time index) in the bit stream $\boldsymbol{u} = (u_1, u_2, ..., u_k, ..., u_N) \in [0,1]^N$ according to its hard value in the BSC as follows:

$$L'(u_k | y_{[j]}) = \alpha . L(u_k | y_{[j]}) + \beta \tag{3.24}$$

Here, $y_{[j]} = (y_{1,j}, y_{2,j}, ..., y_{N,j}, ... y_{2N-1,j}, y_{2N,j}) \in \Re^{2N}$ is the received noisy bit sequence at the jth iteration. $L(u_k | y_{[j]})$ is the soft value of bit u_k obtained after the channel decoding at the jth iteration, defined by Equation 3.2. $L'(u_k | y_{[j]})$ is the modified version of $L(u_k | y_{[j]})$. The values of the two modification parameters α and β are chosen empirically (see Table 3.2) taking into account three conditions: whether the bit is flipped, whether the BSC passes verification, the sign of the bit in question.

Table 3.2: Modification Parameter α and β [5]

Type of Bit	Sign of Bit	BSC passes verification		BSC fails verification	
		α	β	α	β
Flip-Bit	-	0.1	5	1	2.5
	+	0.1	-5	1	-2.5
Non-Flip-Bit	-	1	-5	1	0
	+	1	5	1	0

The modified soft values of the bit stream is interleaved and fed back into the channel decoder for the next iteration as in [5].

3.7 Performance Metrics

As discussed in Chapter 1, PSNR and BER are the two metrics used to evaluate the performance of the proposed schemes. Also, at each channel SNR value, the simulation is run several times and with the average being taken. To investigate the accuracy of the averaged values, the Chebyshev's upper bound and lower bound are plotted. At each channel SNR value, denote:

- M: the number of run of simulations
- X_i: the PSNR value obtained at the i^{th} run
- $\bar{X}$: the averaged PSNR value (obtained after M runs)

The averaged PSNR value is calculated by:

$$\bar{X} = \frac{1}{M}\sum_{i=1}^{M} X_i \tag{3.25}$$

Since X_1 , X_2 .., X_M represent a random sample of size M, the sample standard deviation is calculated as [42]:

$$S_X = \sqrt{\frac{1}{M-1}\sum_{i=1}^{M}(X_i - \bar{X})^2} \tag{3.26}$$

The standard deviation of random variable $\bar{X}$, denoted $S_{\bar{X}}$ is calculated as [42]:

$$S_{\bar{X}} = \frac{S_X}{\sqrt{M}} \tag{3.27}$$

According to Chebyshev's theorem [42], the probability that the estimated mean value $\bar{X}$ will be a value within z standard deviations of the mean is at least $1 - 1/z^2$. With the choice of 90% of confidence (this leads to the choice of $z = \sqrt{10}$), the Chebyshev's upper bound and lower bound are given as follows:

- Chebyshev's upper bound: $\bar{X} + \sqrt{10}S_{\bar{X}}$
- Chebyshev's lower bound: $\bar{X} - \sqrt{10}S_{\bar{X}}$

These two bounds mean that, the averaged PSNR value $\bar{X}$ falls into the range $[\bar{X} - \sqrt{10}S_{\bar{X}} ; \bar{X} + \sqrt{10}S_{\bar{X}}]$ with the probability of at least 90%.

3.8 Simulation Results

To evaluate the performance of the proposed schemes (IJSCD-VC-FS and IJSCD-VC-PI), two sets of simulations are performed, using the two video sequences "Football" and "Table-Tennis".

3.8.1 Simulation Results for Video "Football"

The 4-second video sequence "Football" with frame size of 352x240 pixels is encoded using Joint Model (JM) software version 9.6 [9]. This video is encoded at bit rate 1Mbps and frame rate 30frames/second. It uses a GOP of length 15 with IBBPBBP structure. Each slice in the video sequence contains one row of macroblocks. After compression, without channel noise, the maximum luminance PSNR (Y-PSNR) of the sequence is 28.97 dB (which is determined by the compression). The size of the compressed bit stream is 4333752 bits.

3.8.1.1 The Choice of the Number of Slice Candidates Actually Verified N_{SC}

As discussed in Section 3.2.7.3, N_{SC} - the number of slice candidates actually verified (without using VC) for each slice is chosen empirically. In this section, a simulation is run to investigate the effect of changing N_{SC}. The IJSCD-VC-FS scheme is run for one iteration with different values of N_{SC}: 50, 100, 300, 500. At each channel SNR value, the simulation is run 10 times and with the average being taken. The simulation results are also compared to the results obtained when the received bit stream is channel decoded and decompressed.

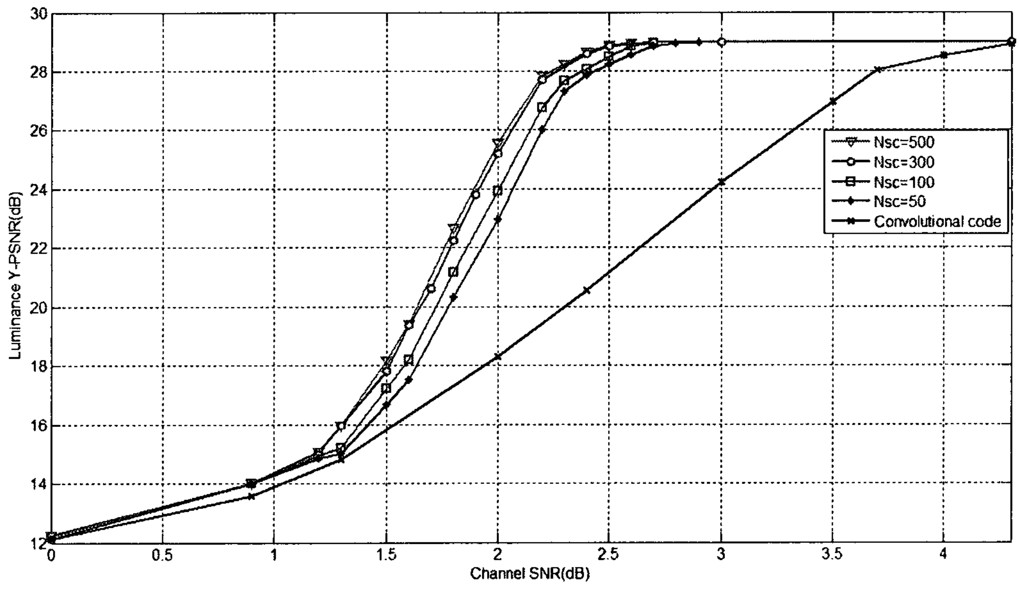


Figure 3.9: Compare the performances with different values of N_{SC}: Y-PSNR vs. channel SNR for video "Football".

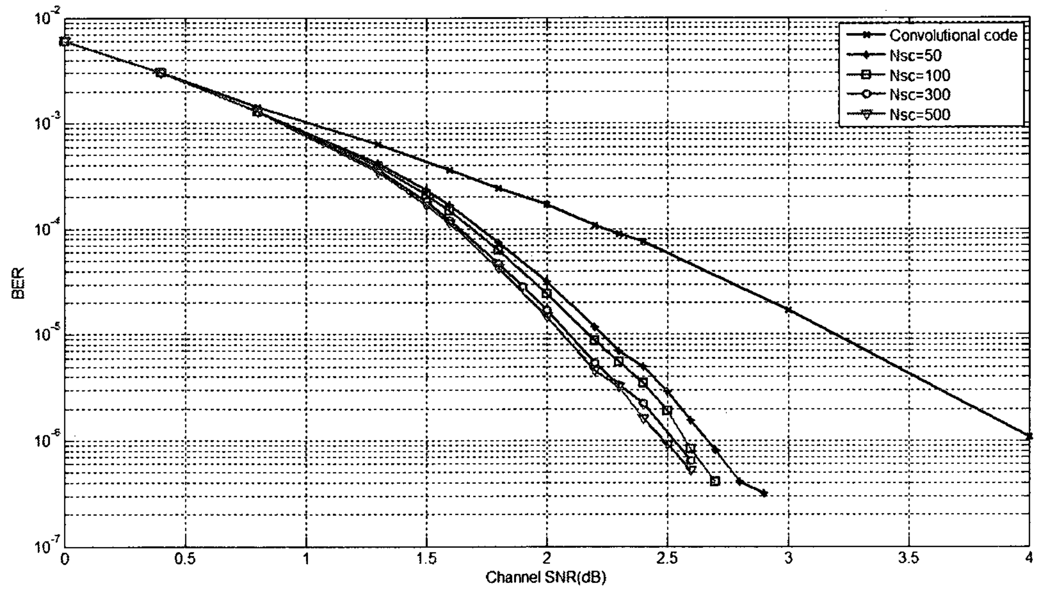


Figure 3.10: Compare the performances with different values of N_{SC} : BER vs. channel SNR for video "Football".

Figure 3.9 and 3.10 show the objective performance in terms of Y-PSNR and BER respectively when different values of N_{SC} are used. With convolutional decoding only, the saturated Y-PSNR is achieved at the channel SNR of 4.3dB. Meanwhile, using 50, 100, 300, 500 slice candidates, the saturated Y-PSNR is achieved at channel SNR of 2.7dB, 2.6dB, 2.5dB and 2.5 dB respectively. In other words, the gains in channel SNR over convolutional decoding are 1.6dB, 1.7dB, 1.8dB and 1.8dB respectively.

One can see that there is improvement in terms of performance when increasing N_{SC} from 50 to 300. However, almost no performance improvement is observed when increasing N_{SC} from 300 to 500. Hence, N_{SC} is chosen to be 300.

3.8.1.2 Evaluate the Complexity Improvement of IJSCD-VC-FS Scheme

In this section, the complexity improvement of IJSCD-VC-FS Scheme is investigated. N_{SC}=300 slice candidates are generated for each slice. The SSV verifies the semantics of these slice candidates using VC and without using VC. The total number of slice candidates that have been verified in the first iteration is measured. At each channel SNR value, the simulation is run 10 times and with the average being taken.

Table 3.3: Complexity comparison: Non-VC vs FS Scheme for sequence "Football" with 10 runs of simulation

EbN0	**Number of candidates verified without using VC**	**Number of candidates verified using VC (FS Scheme)**	**Percentage of candidates eliminated by VC**	
			Mean	**Standard deviation**
1.2	134526	46478	65.45%	0.29%
1.3	112545	37499	66.68%	0.30%
1.5	73611	22856	68.95%	0.33%
1.6	56577	17753	68.62%	0.42%
1.7	42077	13393	68.17%	0.45%
1.8	29818	9658	67.61%	0.49%
1.9	20816	6960	66.56%	0.62%
2.0	13904	4833	65.24%	0.64%
2.2	5726	2259	60.55%	0.9%
2.4	2373	1123	52.66%	1.3%
2.6	826	492	40.36%	3.3%
2.8	385	265	31%	5.5%

Table 3.3 shows the total number of slice candidates verified for video "Football" at different channel SNR values in case VC is not used and in case it is used. One can see that the higher the channel SNR, the smaller the total number of slice candidates verified.

This is because the higher the channel SNR, the less the number of errors that each slice has, thus the BSC for each slice is likely to be found sooner.

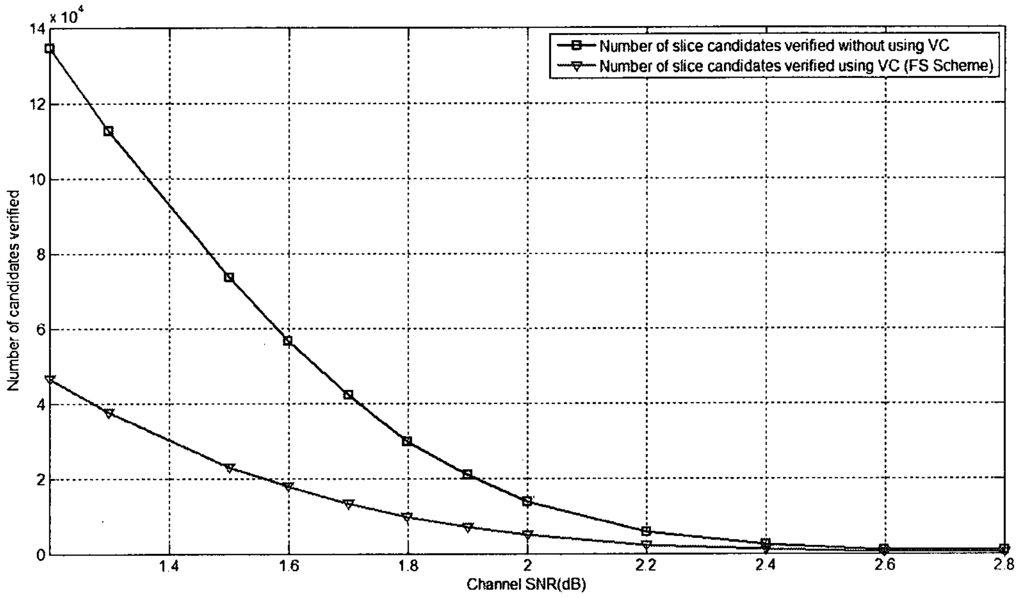


Figure 3.11: Number of slice candidates verified using VC and without using VC for video "Football".

Figure 3.11 visualizes the data shown in Table 3.3. One can see that at low channel SNR values (i.e. lower than 2.2dB), using VC does decrease the number of slice candidates actually verified. However, at higher channel SNR values, the SSV using VC and without using VC has approximately the same complexity. This is because, as discussed above, at high channel SNR values, the SSV finds the BSC quickly and thus the number of slice candidates verified is not significant.

Figure 3.12 compares the complexity of the SSV in terms of time (measured in seconds) using and without using VC. It is observed that at low channel SNR values, using VC does speed up the semantic verification process since the number of slice candidates verified is significant. At high channel SNR values (i.e. at channel SNR higher than 2dB), the number of slice candidates verified is smaller, and hence the amount of time saved for the work of the SSV when using VC is also smaller.

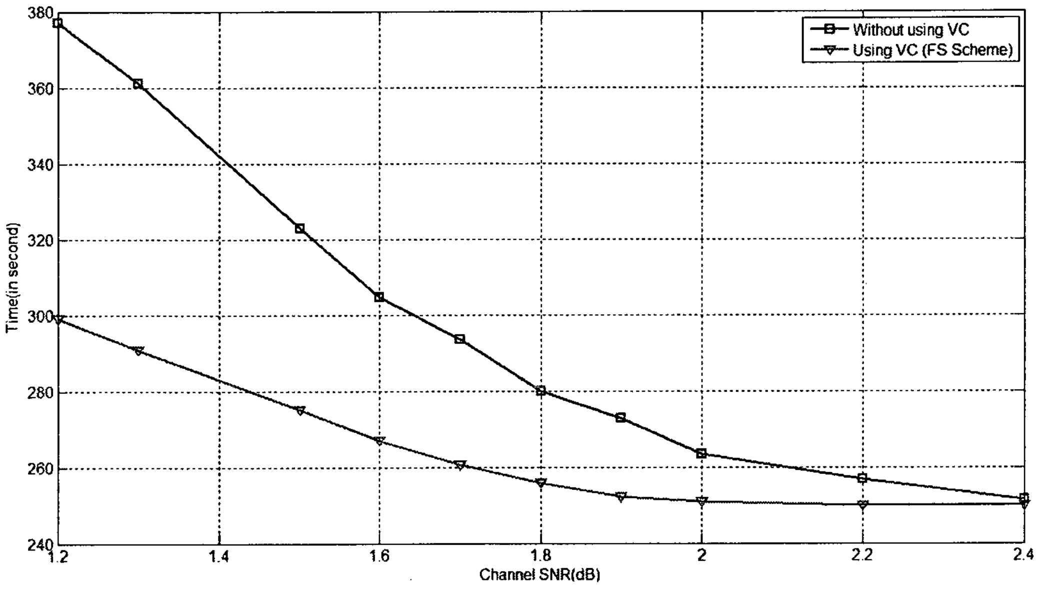


Figure 3.12: Time measured for the work of the SSV with and without using VC for video "Football".

3.8.1.3 Evaluate the Performance of IJSCD-VC-PI Scheme

In this section, simulations are run to compare the performances of the following schemes:

- The scheme proposed by Levine *et.al.* [5]. It is called *the IJSCD-I scheme.*
- The IJSCD-VC-PI scheme.

In the simulations, the number of slice candidates actually verified is N_{SC}=300. The maximum number of slice candidates virtually checked is $N_{MAX} = 2500$. At each channel SNR value, simulations are run 15 times and with the average being taken. Also, in order to investigate the accuracy of the averaged values, the Chebyshev upper bound and lower bound are plotted with 90% of confidence.

3.8.1.3.1 Simulation Results for the IJSCD-I Scheme

Figure 3.13 and Figure 3.14 show the objective performance of the IJSCD-I scheme in terms of PSNR and BER. Results are also compared to the results of the convolutional decoding scheme. One can see that after 4 iterations there is little improvement, particularly in the high PSNR area of the curve. This suggests that further iterations may not be useful.

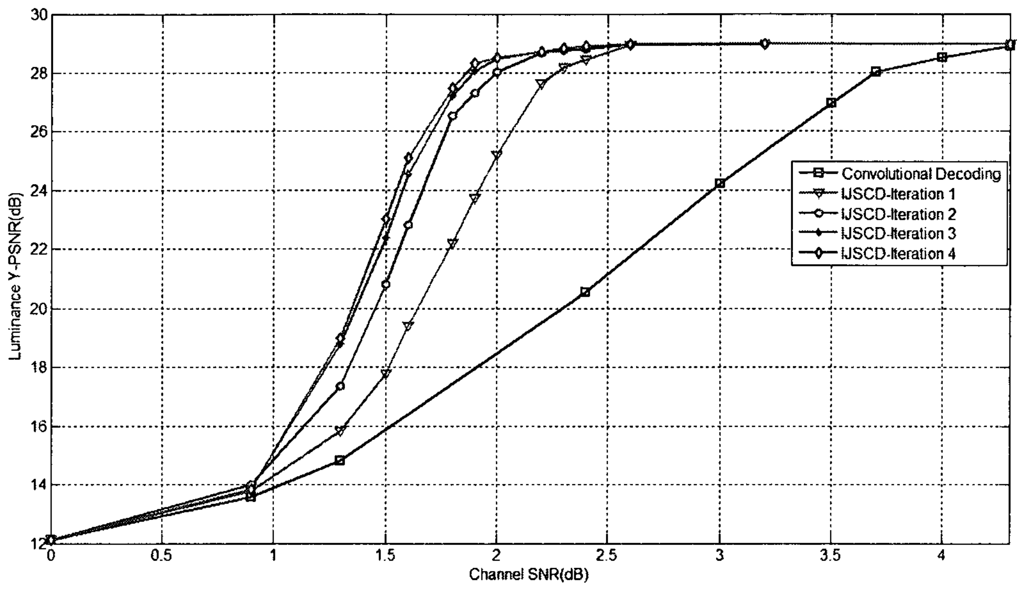


Figure 3.13: Y-PSNR vs. channel SNR for video "Football" of the IJSCD-I scheme.

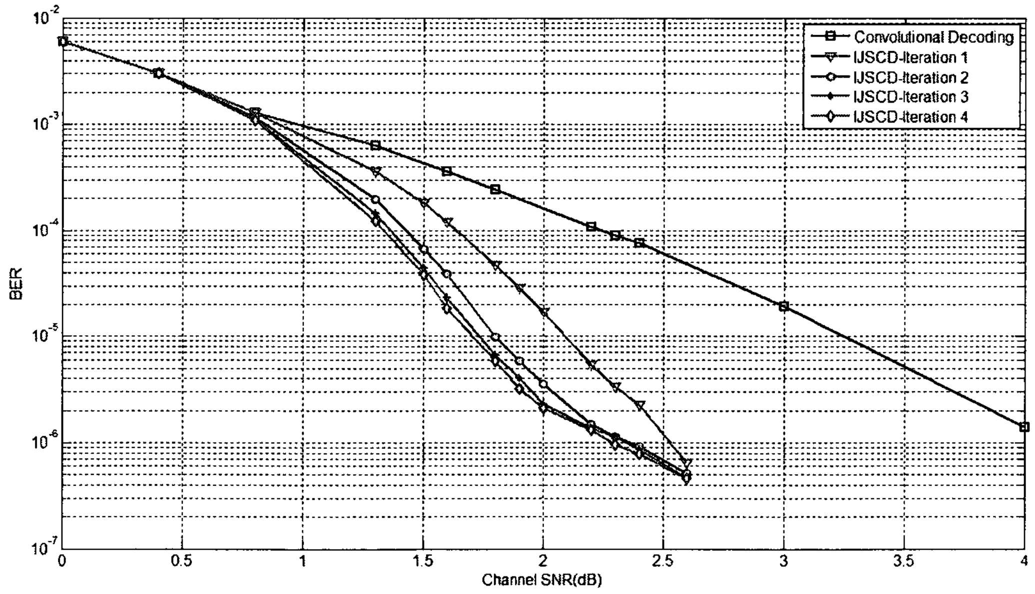


Figure 3.14: BER vs. channel SNR for video "Football" of the IJSCD-I scheme.

Figure 3.15 shows the Chebyshev upper bound and lower bound with 90% of confidence for the Y-PSNR vs. channel SNR curve of the scheme at the 4th iteration.

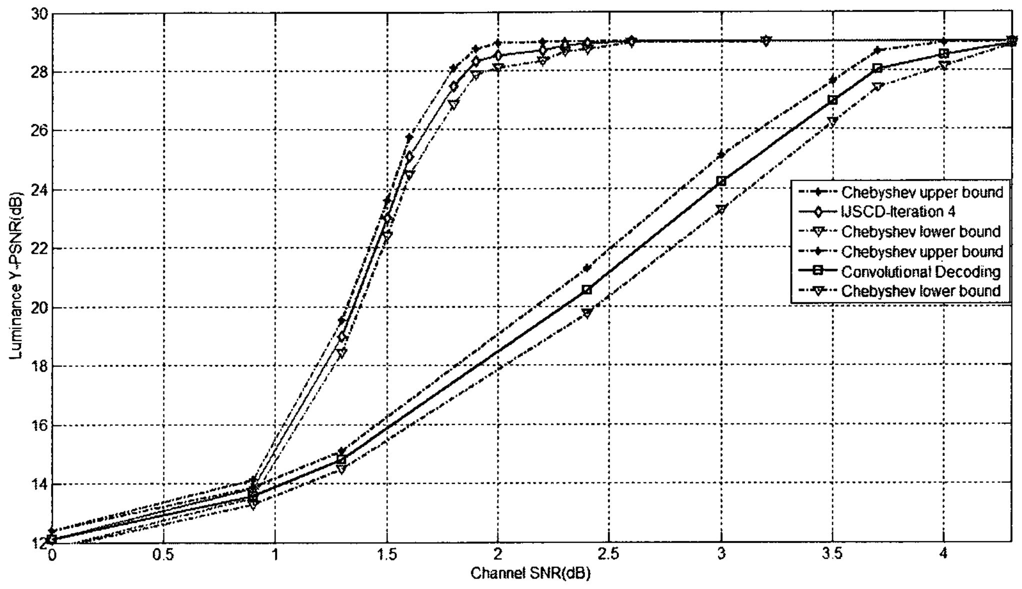


Figure 3.15: Chebyshev's upper bound and lower bound for Y-PSNR vs. channel SNR curves at the 4th iteration of the IJSCD-I scheme and at the output of the channel decoder for video "Football".

3.8.1.3.2 Simulation Results for the IJSCD-VC-PI Scheme

Figure 3.16 and 3.17 show the objective performance of the IJSCD-VC-PI scheme, which is performed up to the fourth iteration since there's almost no improvement after 4 iterations. Results are also compared to the ones of the convolutional coding scheme.

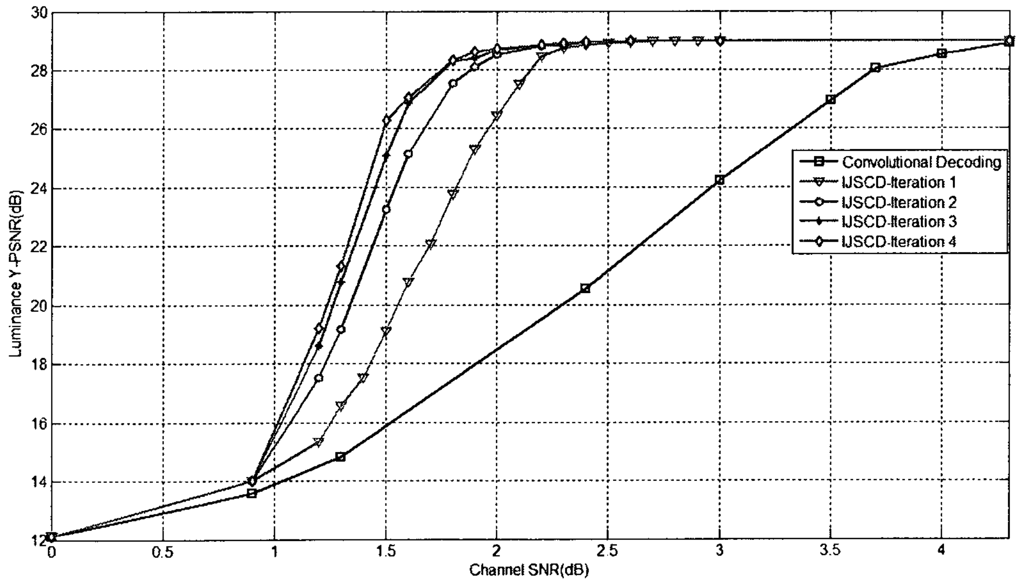


Figure 3.16: Y-PSNR vs. channel SNR for video "Football" of the IJSCD-VC-PI scheme.

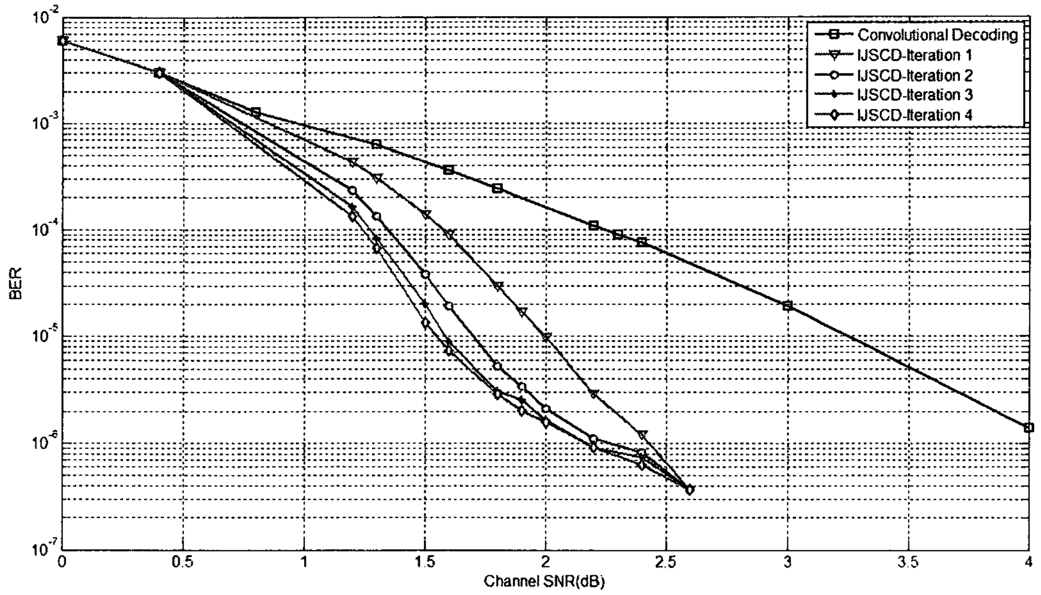


Figure 3.17: BER vs. channel SNR for video "Football" of the IJSCD-VC-PI scheme.

Figure 3.18 shows the Chebyshev upper bound and lower for the Y-PSNR vs. channel SNR curve at the 4th iteration.

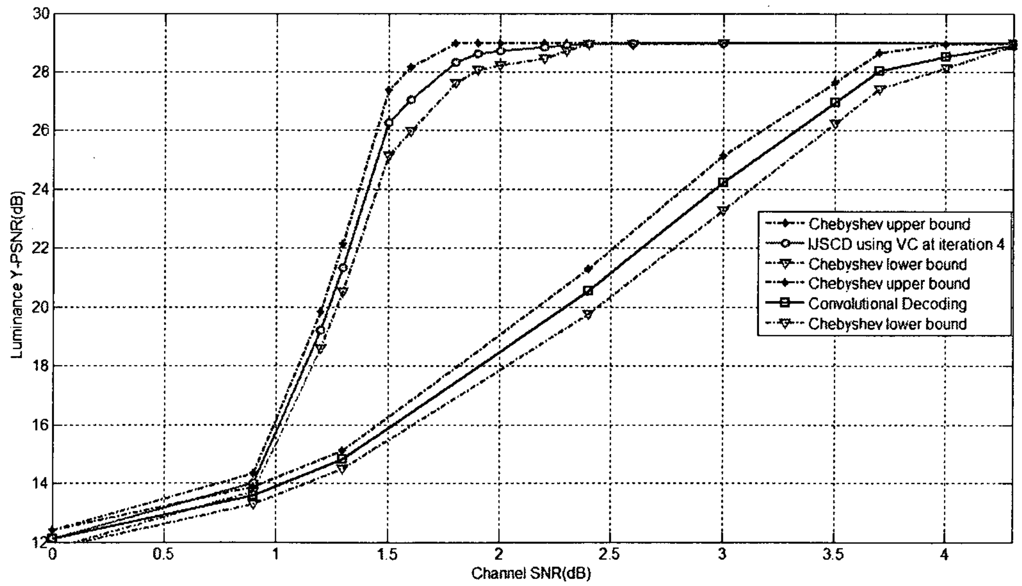


Figure 3.18: Chebyshev upper bound and lower bound for Y-PSNR vs. channel SNR curves at the 4th iteration of the IJSCD-VC-PI scheme and at the output of the channel decoder for video "Football".

3.8.1.3.3 *Compare Performance of Schemes at the 4th iteration*

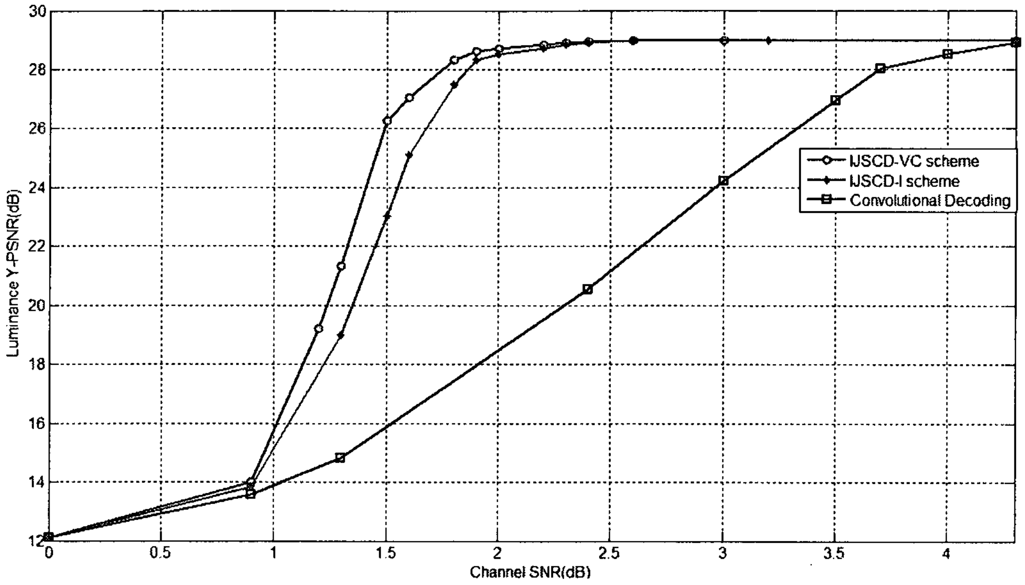


Figure 3.19: Comparison of IJSCD-I scheme and IJSCD-VC-PI scheme: PSNR vs. channel SNR obtained at the 4th iteration for video "Football".

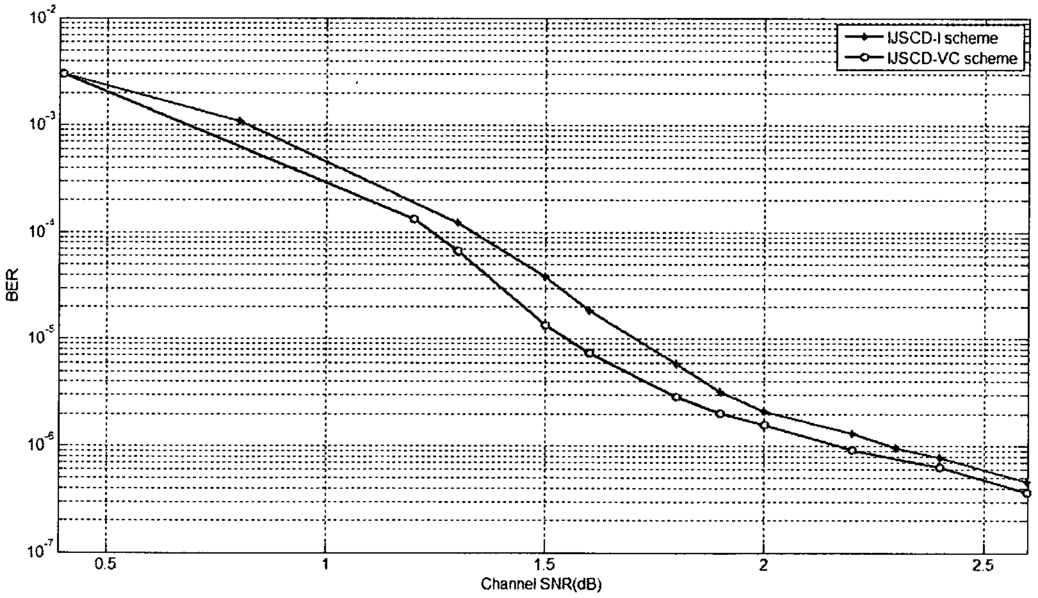


Figure 3.20: Comparison of IJSCD-I scheme and IJSCD-VC-PI scheme: BER vs. channel SNR obtained at the 4th iteration for video "Football".

Figure 3.19 and 3.20 compare the performance of the two schemes at the 4th iteration. With convolutional decoding, the maximum achievable PSNR is obtained at channel

SNR of 4.3dB. The IJSCD-I scheme obtains that value at channel SNR of 2.2dB. Meanwhile, the IJSCD-VC-PI scheme achieves that value at channel SNR of 2.0dB. Thus, without bandwidth expansion, the IJSCD-VC-PI scheme can achieve 0.2dB of channel SNR reduction over the IJSCD-I scheme, and up to 2.3dB of channel SNR reduction over convolutional decoding.

In terms of computational complexity, generating 2500 slice candidates for every slice in the video sequence takes about 24 seconds. Meanwhile, generating 300 slice candidates for every slice in the same sequence takes about 4 seconds. Hence, the IJSCD-VC-PI scheme keeps almost the same computational complexity when compared to the IJSCD-I scheme proposed in [5].

For the reader's own assessment, the output video of the IJSCD-VC-PI scheme is compared to the one of the IJSCD-I scheme and of the convolutional decoding. Frames 31 and 64 of the output videos of these schemes at channel SNR of 1.8dB are shown.

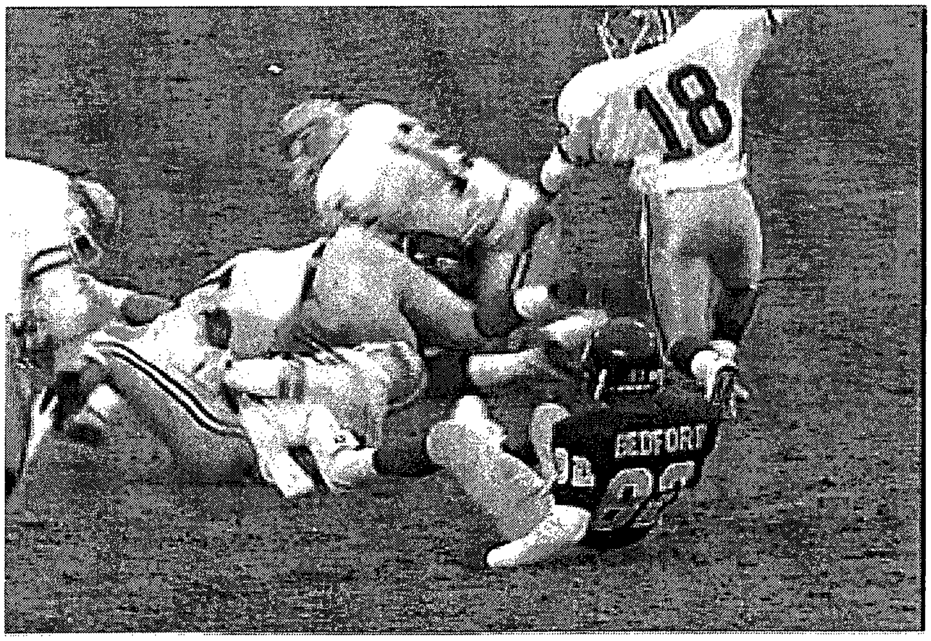


Figure 3.21: Decompressed error-free frame 31 of video sequence "Football".

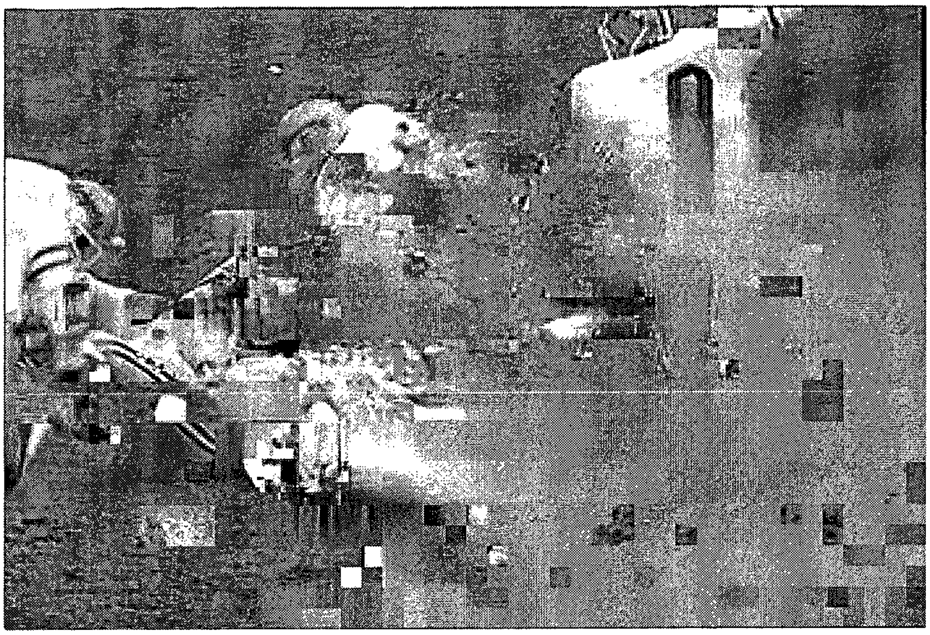


Figure 3.22: Frame 31 of video sequence “Football” transmitted over an AWGN channel at channel SNR of 1.8dB using convolutional decoding only.

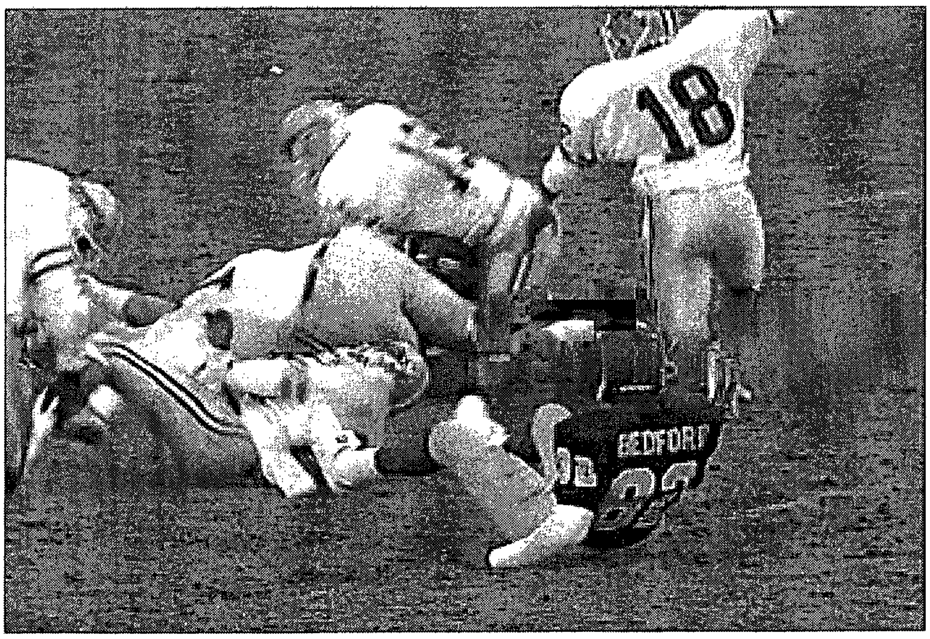


Figure 3.23: Frame 31 of video sequence “Football” decoded by the IJSCD-I scheme at channel SNR of 1.8dB and at the 4th iteration.

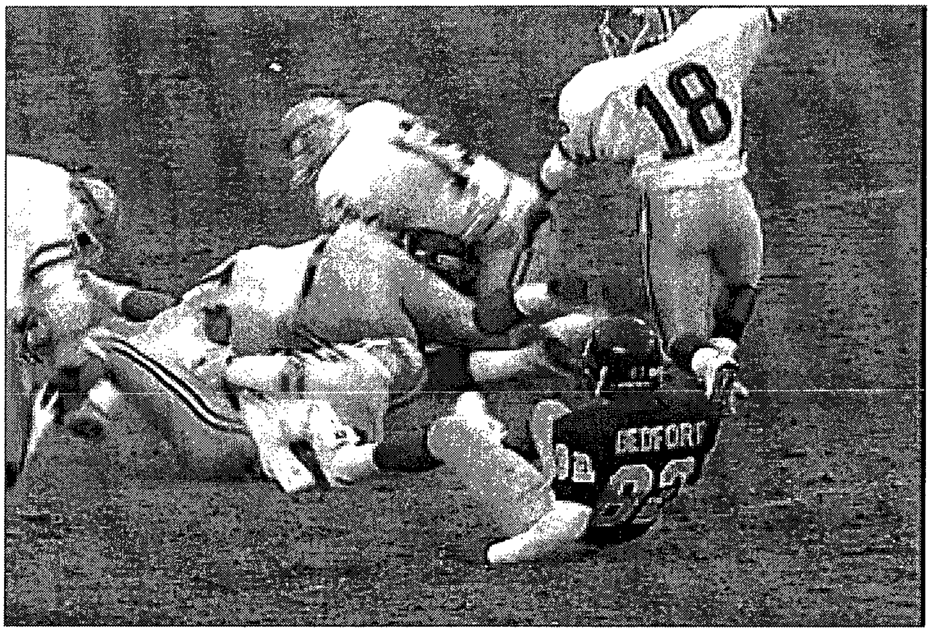


Figure 3.24: Frame 31 of video sequence "Football" decoded by the IJSCD-VC-PI scheme at channel SNR of 1.8dB and at the 4th iteration.

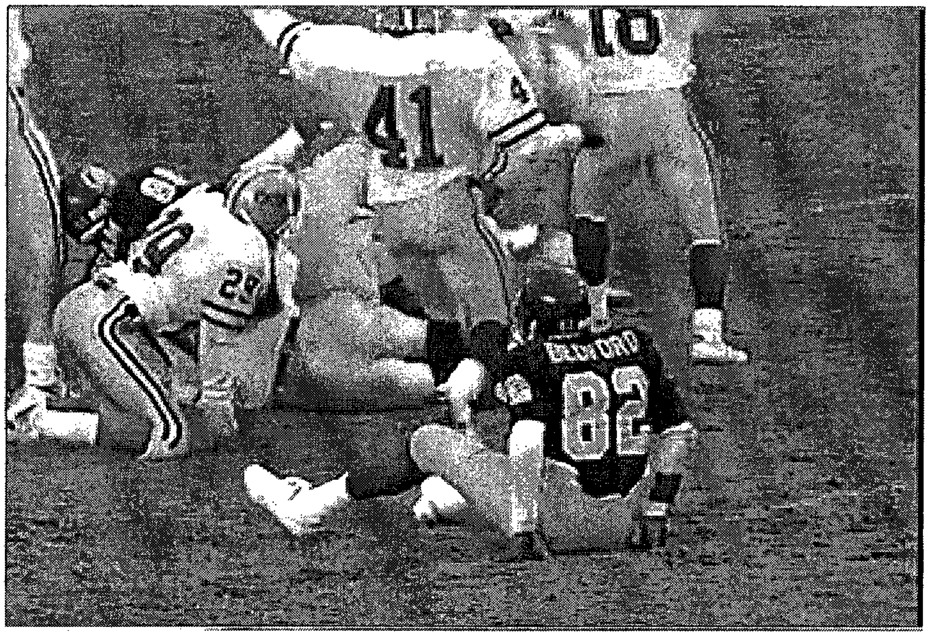


Figure 3.25: Decompressed error-free frame 64 of video sequence "Football".

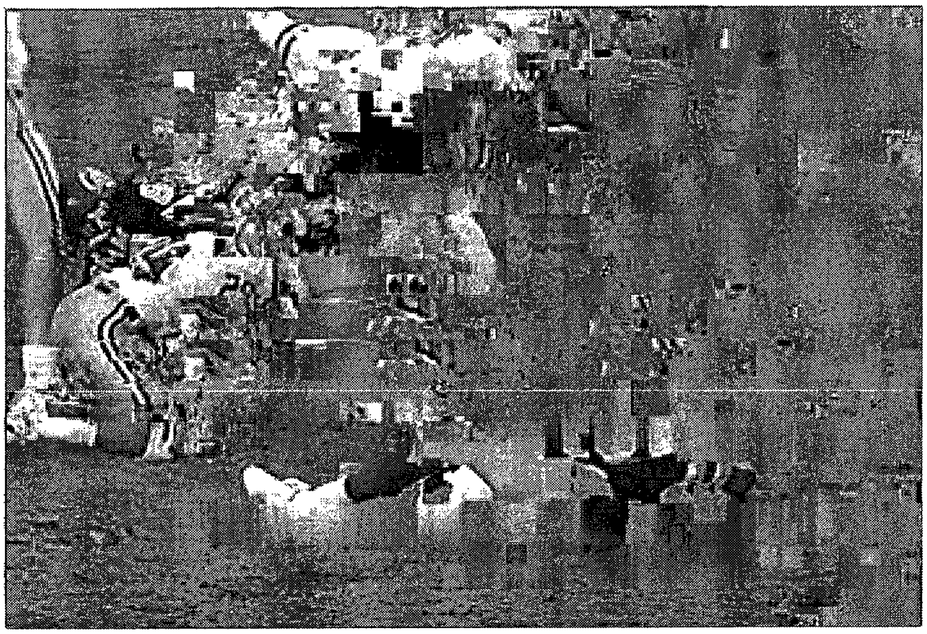


Figure 3.26: Frame 64 of video sequence "Football" transmitted over an AWGN channel at channel SNR of 1.8dB using convolutional decoding only.

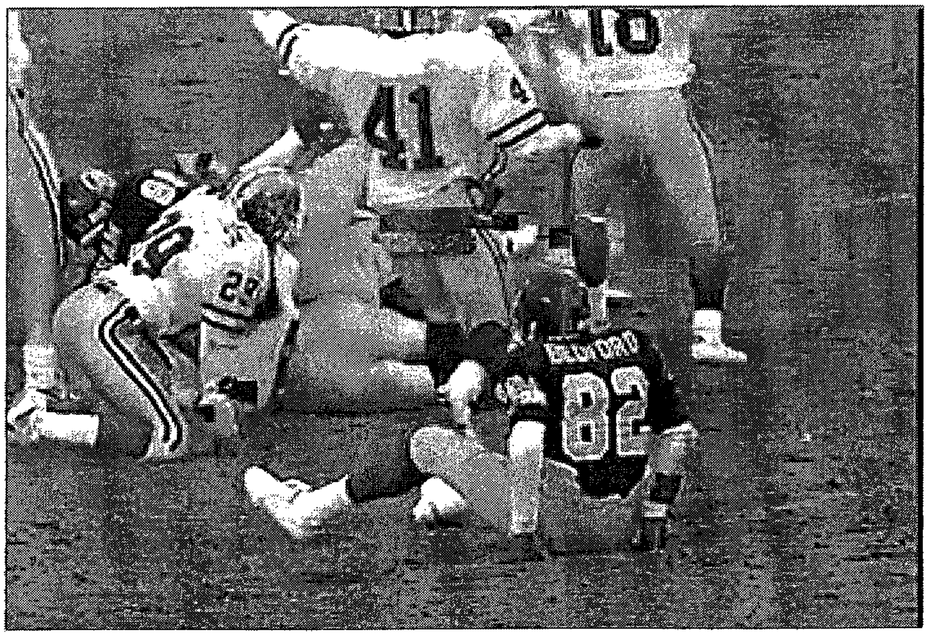


Figure 3.27: Frame 64 of video sequence "Football" decoded by the IJSCD-I scheme at channel SNR of 1.8dB and at the 4th iteration.

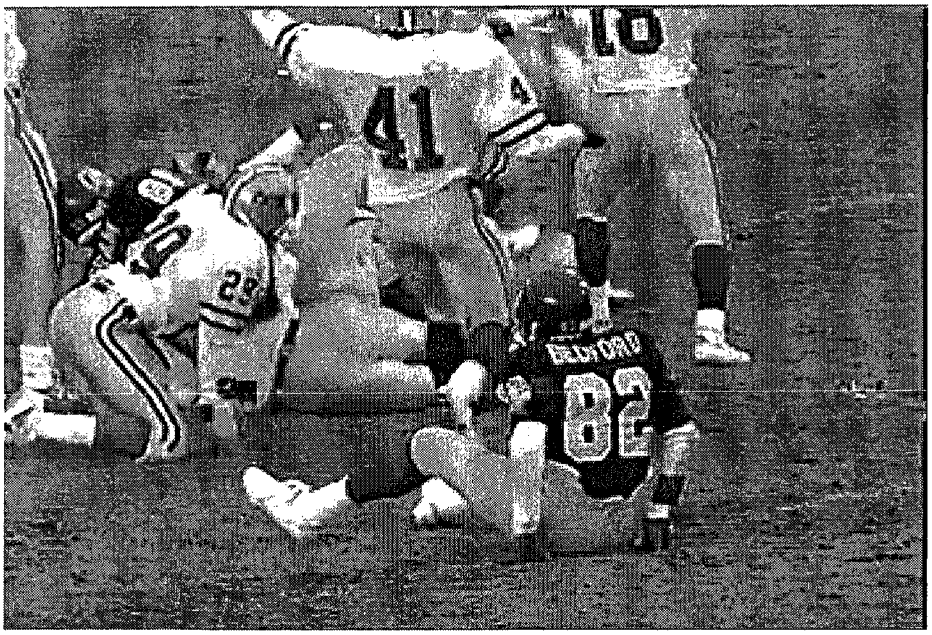


Figure 3.28: Frame 64 of video sequence "Football" decoded by the IJSCD-VC-PI scheme at channel SNR of 1.8dB and at the 4th iteration.

It is observed that the decompressed video using convolutional decoding only is blocky and a large portion of each frame is damaged by black stripes. In contrast, the decompressed video using the IJSCD-I scheme [5] is mostly viewable although some frames still have visible errors. The visual quality of the decompressed video using IJSCD-VC-PI scheme is further improved with less visible errors.

3.8.2 Simulation Results for Video "Table-Tennis"

The 110-frame video sequence "Table-Tennis" is encoded using the same parameters used for "Football". After compression, the maximum luminance PSNR of the sequence is 34.35 dB, which is determined by the compression. The size of the compressed bit stream is 3438208 bits.

3.8.2.1 The Choice of the Number of Slice Candidates Actually Verified N_{SC}

Similar to what has been done for "Football", in this section the choice of N_{SC} for video sequence "Table-Tennis" is determined by simulation. The IJSCD-VC-FS scheme is run for one iteration with different values of N_{SC}: 50, 100, 300, 500. At each channel SNR value, the simulation is run 10 times and with the average being taken.

Figure 3.29 and 3.30 show the objective performance in terms of Y-PSNR and BER when different values of N_{SC} are used.

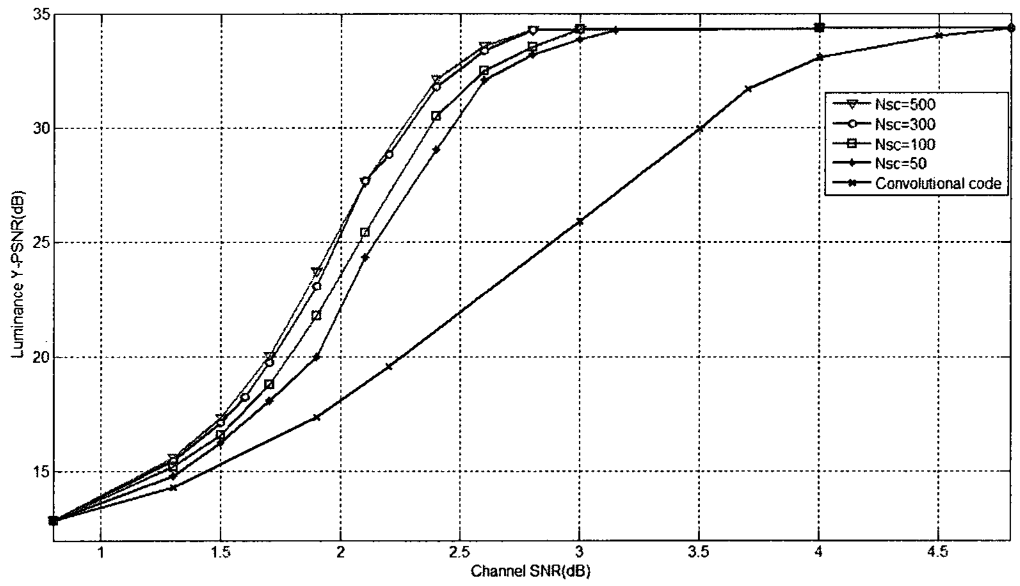


Figure 3.29: Compare the performance with different values of N_{SC}: Y-PSNR vs. channel SNR for video "Table-Tennis".

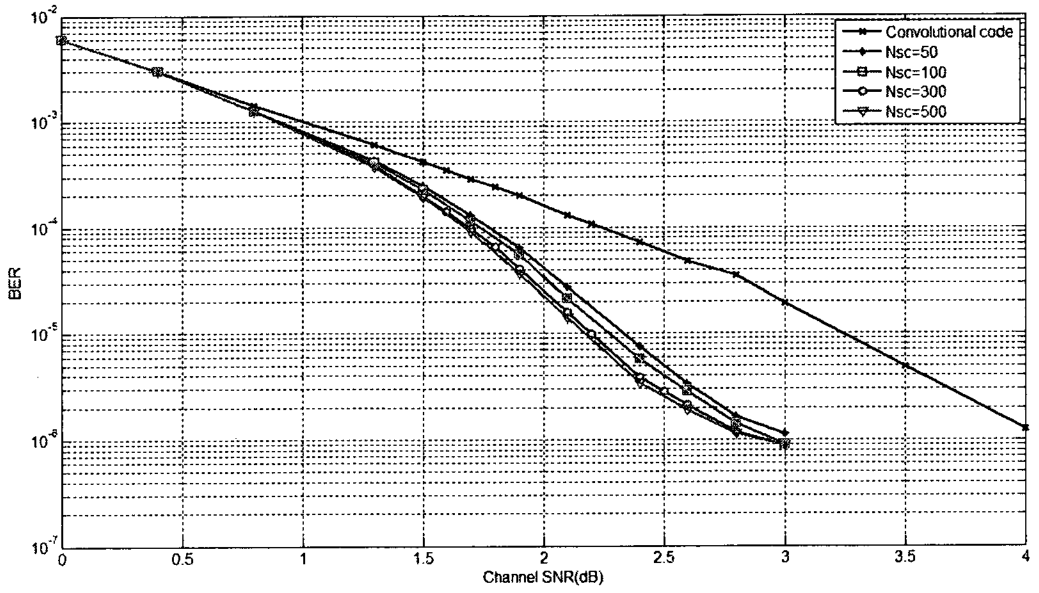


Figure 3.30: Compare the performance with different values of N_{SC} : BER vs. channel SNR for video "Table-Tennis".

The convolutional decoding scheme achieves the saturated Y-PSNR at channel SNR of 4.8dB. Meanwhile, using 50, 100, 300, 500 slice candidates, the saturated Y-PSNR is achieved at channel SNR of 3.1dB, 3.0dB, 2.8dB and 2.8dB respectively. In other words, the gains in channel SNR over convolutional decoding are 1.7dB, 1.8dB, 2.0dB and 2.0dB respectively. One can see that almost no performance improvement is observed when increasing N_{SC} from 300 to 500. Hence, N_{SC} is chosen to be 300.

3.8.2.2 Evaluate the Complexity Improvement of IJSCD-VC-FS Scheme

In this section, the complexity improvement of IJSCD-VC-FS Scheme is investigated for "Table-Tennis". N_{SC}=300 slice candidates are generated for each slice. The SSV verifies the semantics of these slice candidates using VC and without using VC. The total number of slice candidates that have been verified in the first iteration is measured. At each channel SNR value, the simulation is run 10 times and with the average being taken.

Table 3.4: Complexity comparison: Non-VC vs FS Scheme for sequence "Table-Tennis" with 10 runs of simulation

EbN0	**Number of candidates verified without using VC**	**Number of candidates verified using VC (FS Scheme)**	**Percentage of candidates eliminated by VC**	
			Mean	**Standard deviation**
1.5	55229	20826	62.29%	0.52%
1.6	44818	16210	63.83%	0.36%
1.7	35277	12269	65.22%	0.85%
1.9	19689	6456	67.21%	1.17%
2.0	14179	4693	66.9%	1.27%
2.2	6591	2240	66%	1.63%
2.4	2853	1132	60.3%	2.76%
2.6	1410	665	52.8%	2.87%
2.8	819	468	42.74%	2.4%

Table 3.4 shows the number of slice candidates verified by the SSV using VC and without using VC. It also shows the percentage of slice candidates eliminated by VC.

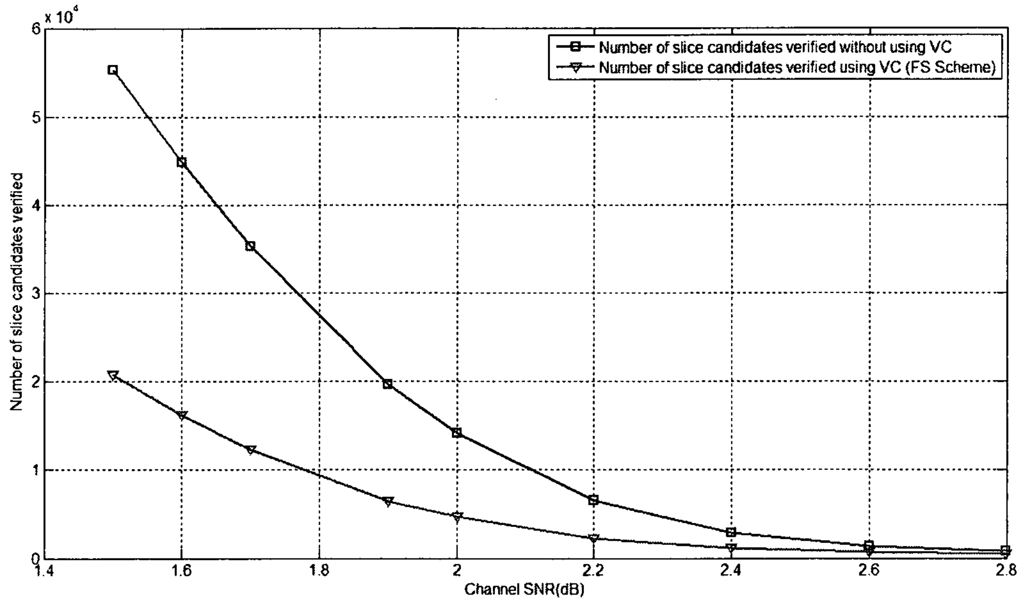


Figure 3.31: Number of slice candidates verified using and without using VC for video "Table-Tennis".

Figure 3.31 visualizes the data shown in Table 3.4. It is observed that the number of slice candidates verified using VC is smaller than the one without using VC, particularly at channel SNR lower than 2.2dB. This is because at low channel SNR, each slice has more errors. Hence, the SSV has to verify many slice candidates to find the BSCs. Therefore, using VC in this case can eliminate many slice candidates. At channel SNR higher than 2.4dB, using VC and without using VC offer almost the same complexity since the number of slice candidates verified is not significant.

Figure 3.32 compares the complexity of the SSV in terms of time (measured in seconds) using VC and without using VC. As what was discussed for video "Football", one can see that at low channel SNR values, using VC does speed up the semantic verification process since the number of slice candidates verified is significant. When channel SNR increases, the number of slice candidates verified is smaller, and hence the amount of time saved for the work of the SSV using VC is also smaller.

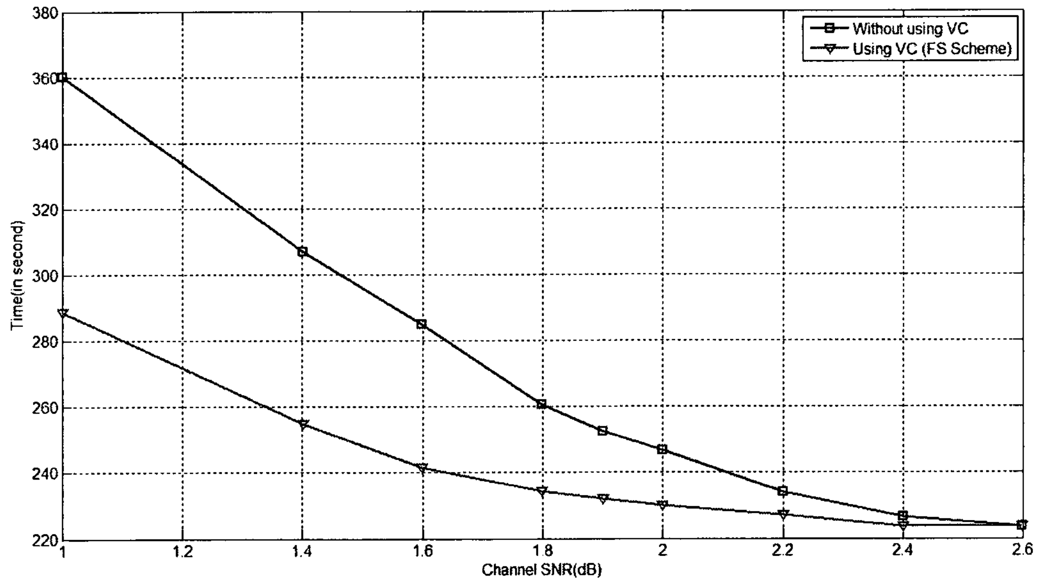


Figure 3.32: Time measured for the work of the SSV with and without using VC for video "Table-tennis".

3.8.2.3 Evaluate the Performance of IJSCD-VC-PI Scheme

Similar to the simulations performed for "Football", in this section, simulation is run to compare the performances of the IJSCD-I scheme and the IJSCD-VC-PI scheme. The number of slice candidates actually verified is $N_{SC} = 300$. The maximum number of slice candidates virtually checked is $N_{MAX} = 2500$. At each channel SNR value, the simulation is run 15 times and with the average being taken. Also, the Chebyshev upper bound and lower bound with 90% of confidence are plotted.

3.8.2.3.1 Simulation Results for the IJSCD-I Scheme

Figure 3.33 and Figure 3.34 show the objective performance of the IJSCD-I scheme in terms of PSNR and BER. Results are also compared to the results of the convolutional decoding scheme. The scheme is run with 4 iterations since no improvement is observed after 4 iterations. The Chebyshev upper bound and lower bound is shown in Figure 3.35.

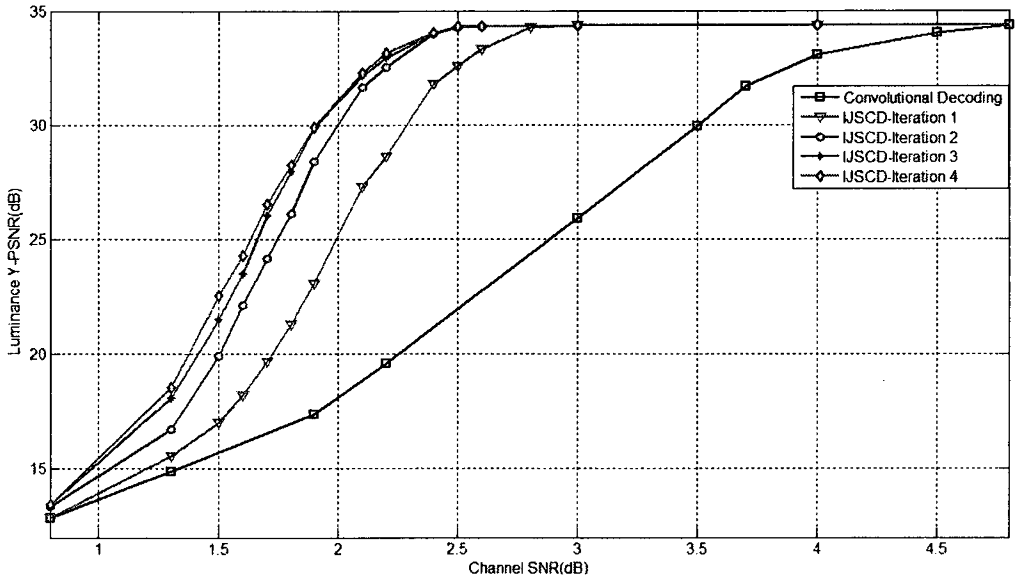


Figure 3.33: Y-PSNR vs. channel SNR for video "Table-Tennis" of the IJSCD-I scheme.

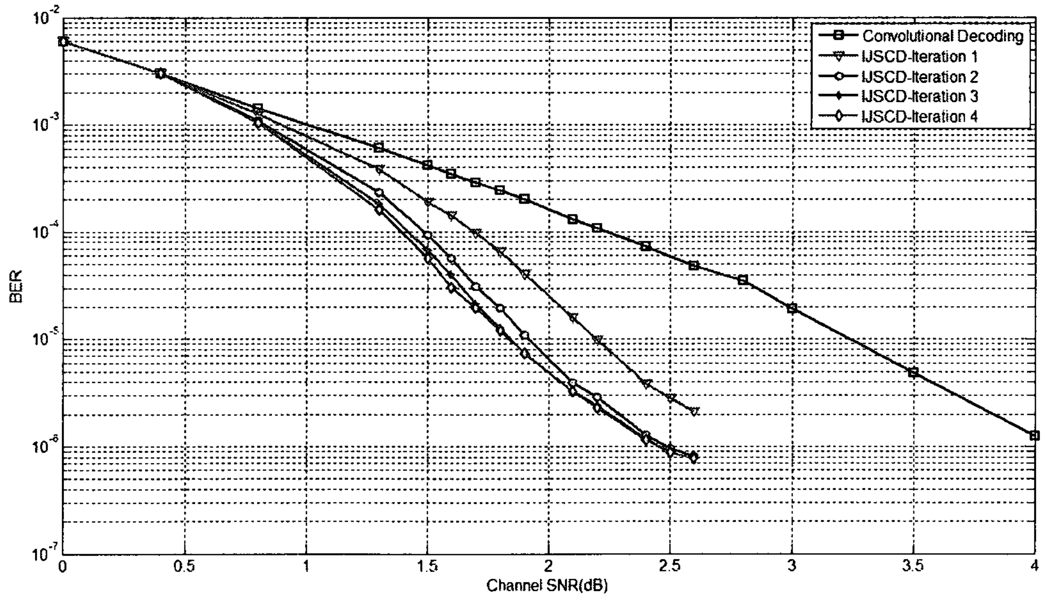


Figure 3.34: BER vs. channel SNR for video "Table-Tennis" of the IJSCD-I scheme.

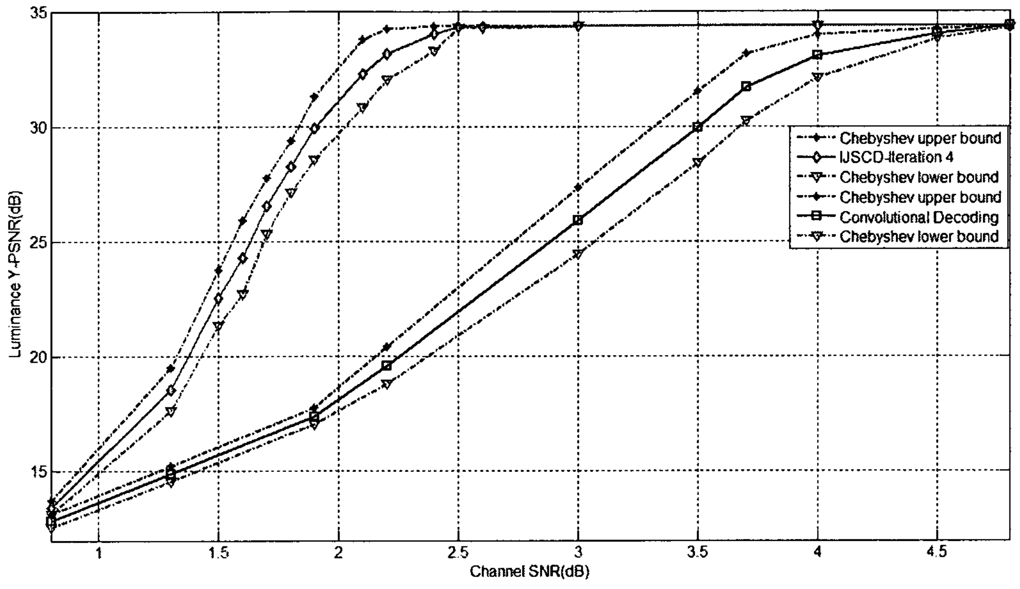


Figure 3.35: Chebyshev's upper bound and lower bound for Y-PSNR vs. channel SNR curves at the 4th iteration of the IJSCD-I scheme and at the output of the channel decoder for video "Table-Tennis".

3.8.2.3.2 Simulation Results for the IJSCD-VC-PI Scheme

Figure 3.36 and 3.37 show the objective performance (PSNR and BER) for the IJSCD-VC-PI scheme. The Chebyshev upper bound and lower bound is shown in Figure 3.38.

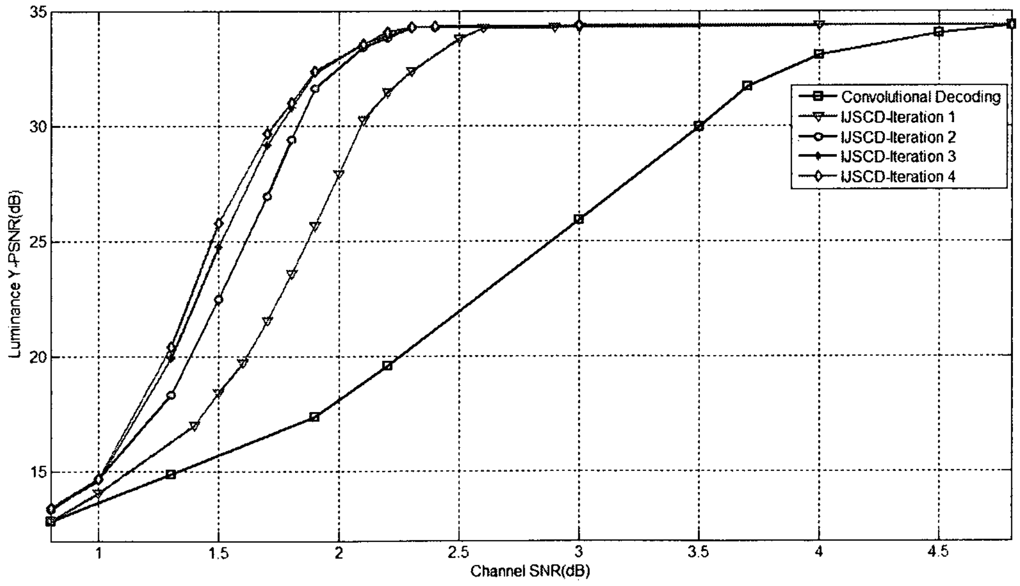


Figure 3.36: Y-PSNR vs. channel SNR for video "Table-Tennis" of the IJSCD-VC-PI scheme.

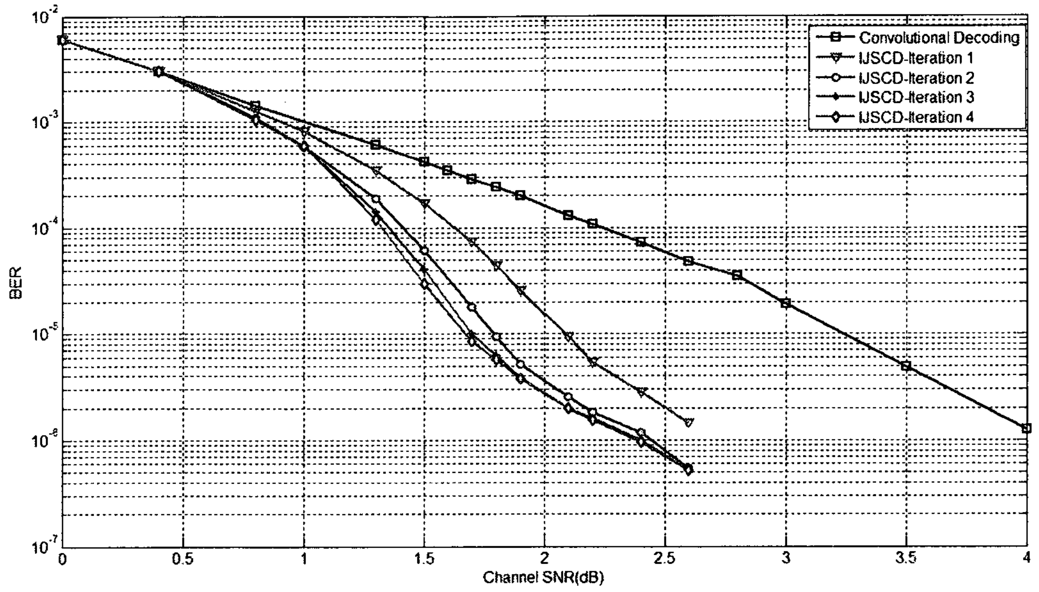


Figure 3.37: BER vs. channel SNR for video "Table-Tennis" of the IJSCD-VC-PI scheme.

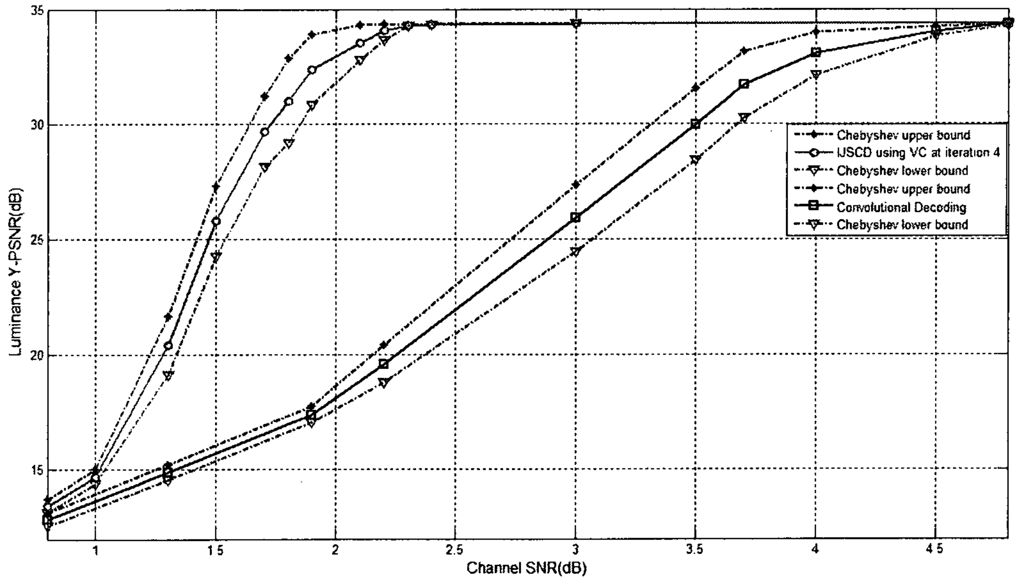


Figure 3.38: Chebyshev's upper bound and lower bound for SNR vs. channel SNR curves at the 4th iteration of the IJSCD-VC-PI scheme and at the output of the channel decoder for video "Table-Tennis".

3.8.2.3.3 Compare Performance of Schemes at the 4th iteration

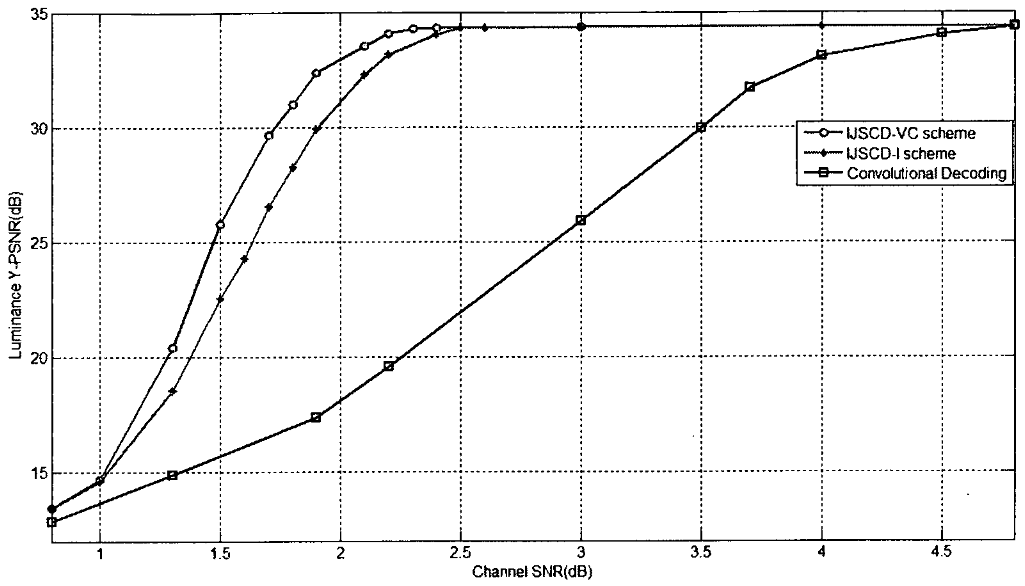


Figure 3.39: Comparison of IJSCD-I scheme and IJSCD-VC-PI scheme: PSNR vs. channel SNR obtained at the 4th iteration for video "Table-Tennis".

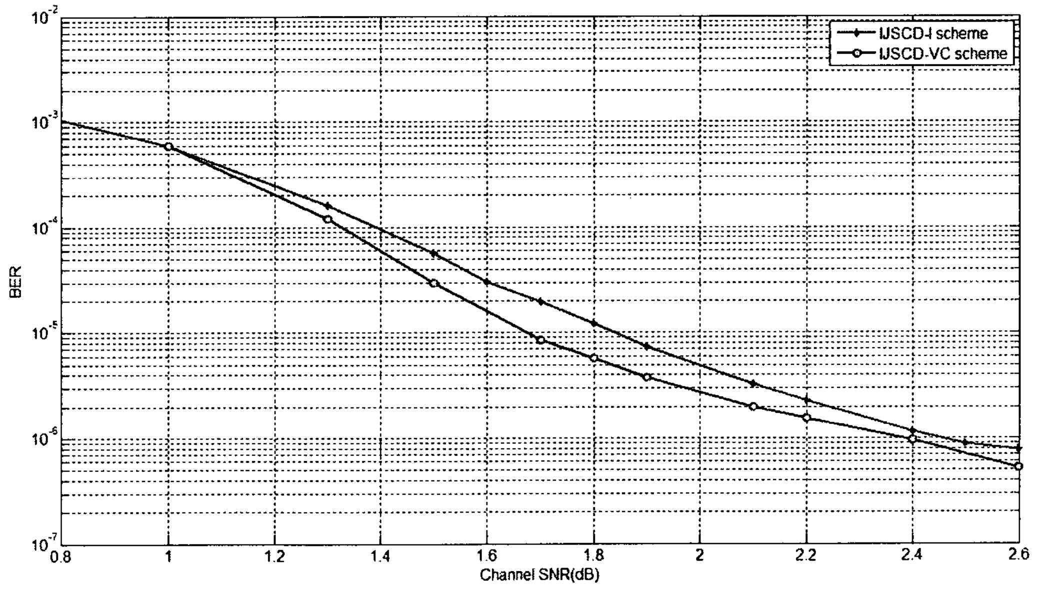


Figure 3.40: Comparison of IJSCD-I scheme and IJSCD-VC-PI scheme: BER vs. channel SNR obtained at the 4th iteration for video "Table-Tennis".

Figure 3.39 and 3.40 compare the performance of the two schemes at the 4th iteration. With convolutional decoding, the maximum achievable PSNR is obtained at channel SNR of 4.8dB. The IJSCD-I scheme obtains that value at channel SNR of 2.5dB.

Meanwhile, the IJSCD-VC-PI scheme achieves that value at the channel SNR of 2.3dB. Thus, without bandwidth expansion, the IJSCD-VC-PI scheme can achieve 0.2dB of channel SNR reduction over the IJSCD-I scheme (with almost the same computational complexity), and up to 2.5dB of channel SNR reduction over convolutional decoding.

For the reader's own assessment, frames 55 and 100 of the output videos of these schemes at channel SNR of 1.7dB are shown below. One can see that there are improvements in terms of picture quality when the IJSCD-VC-PI scheme is used in comparison to the IJSCD-I scheme and convolutional decoding scheme.

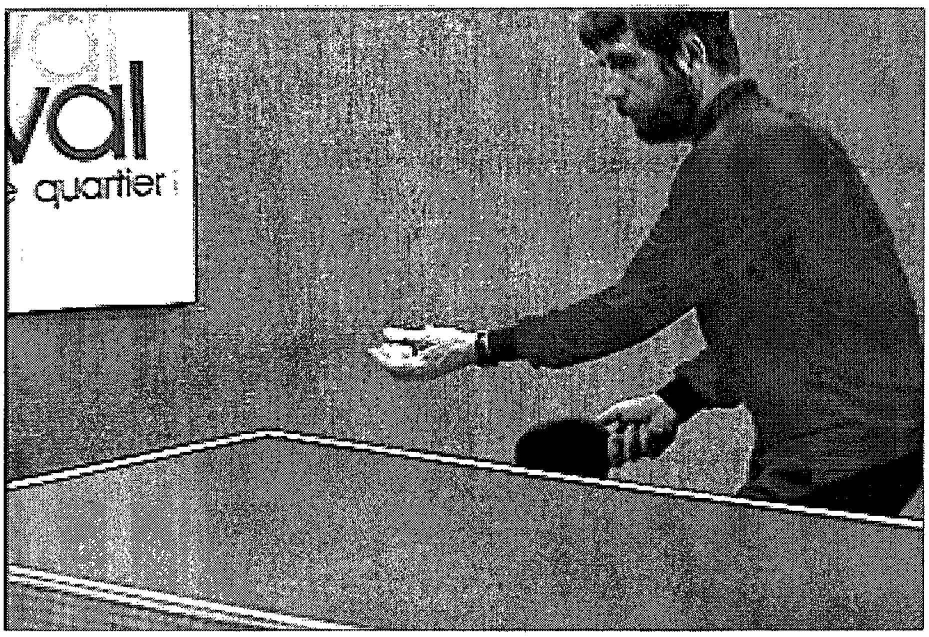


Figure 3.41: Decompressed error-free frame 55 of video sequence "Table-Tennis".

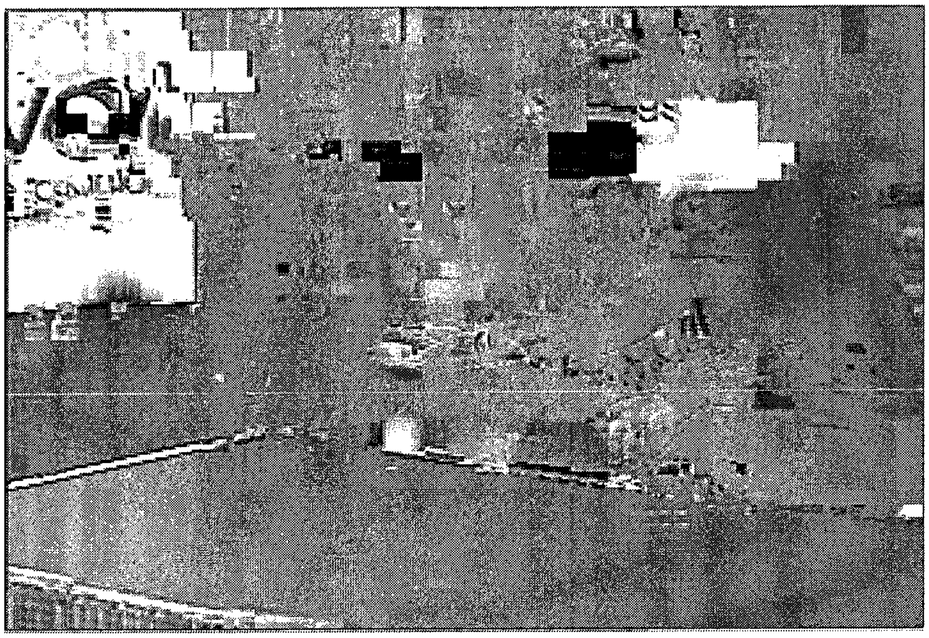


Figure 3.42: Frame 55 of video sequence “Table-Tennis” transmitted over an AWGN channel at channel SNR of 1.7dB using convolutional decoding only.

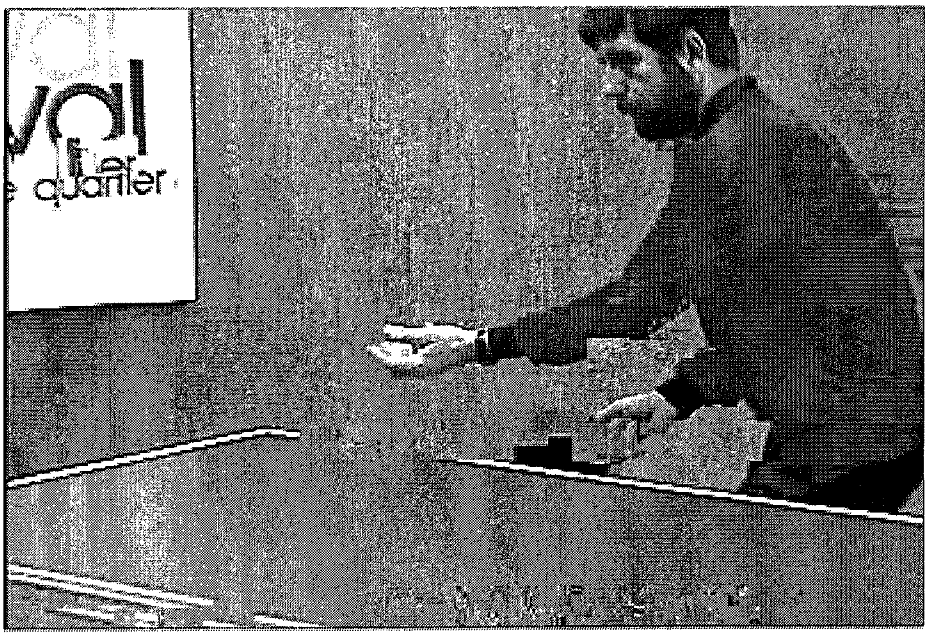


Figure 3.43: Frame 55 of video sequence “Table-Tennis” decoded by the IJSCD-I scheme at channel SNR of 1.7dB and at the 4th iteration.

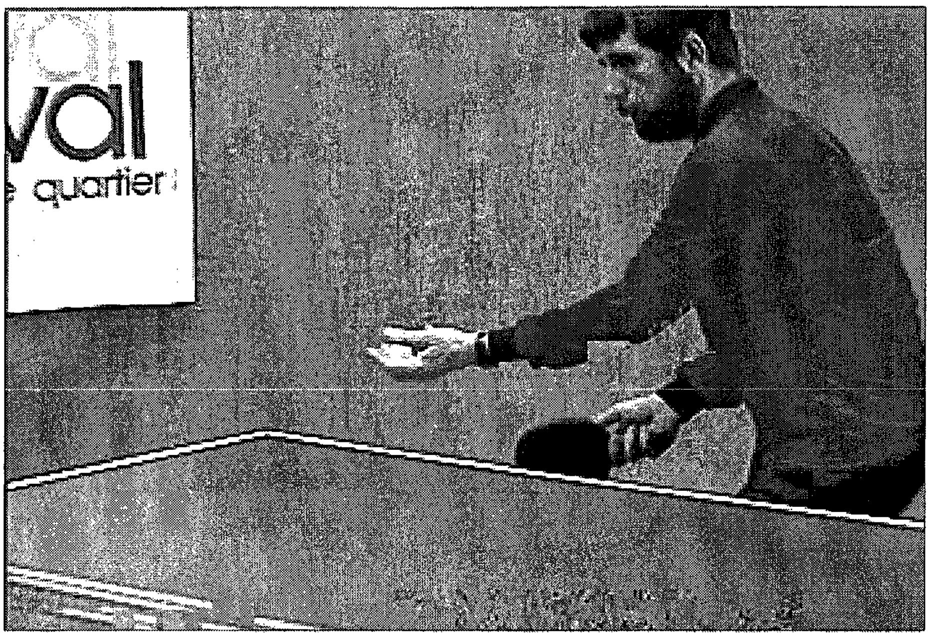


Figure 3.44: Frame 55 of video sequence "Table-Tennis" decoded by the IJSCD-VC-PI scheme at channel SNR of 1.7dB and at the 4th iteration.

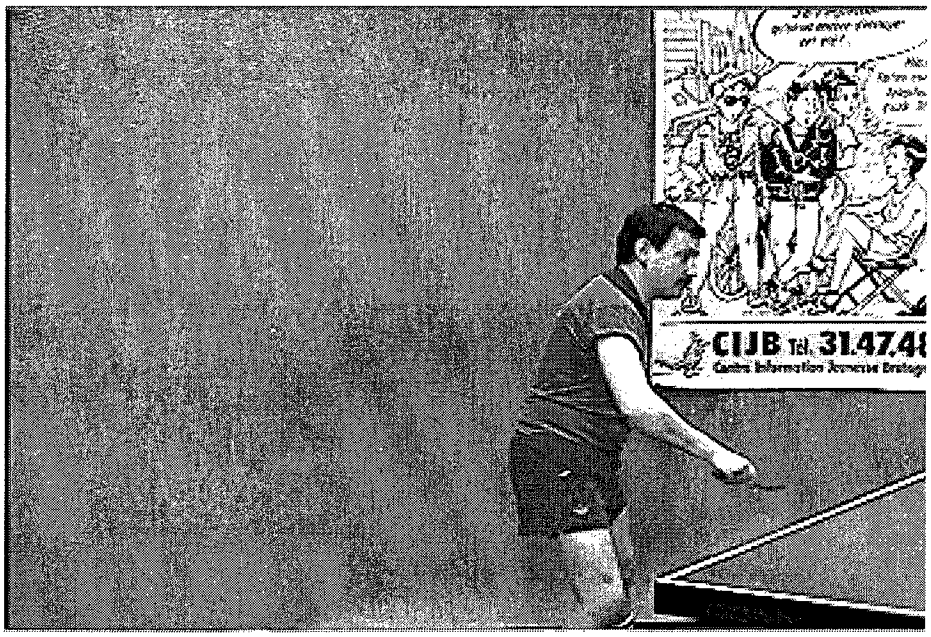


Figure 3.45: Decompressed error-free frame 100 of video sequence "Table-Tennis".

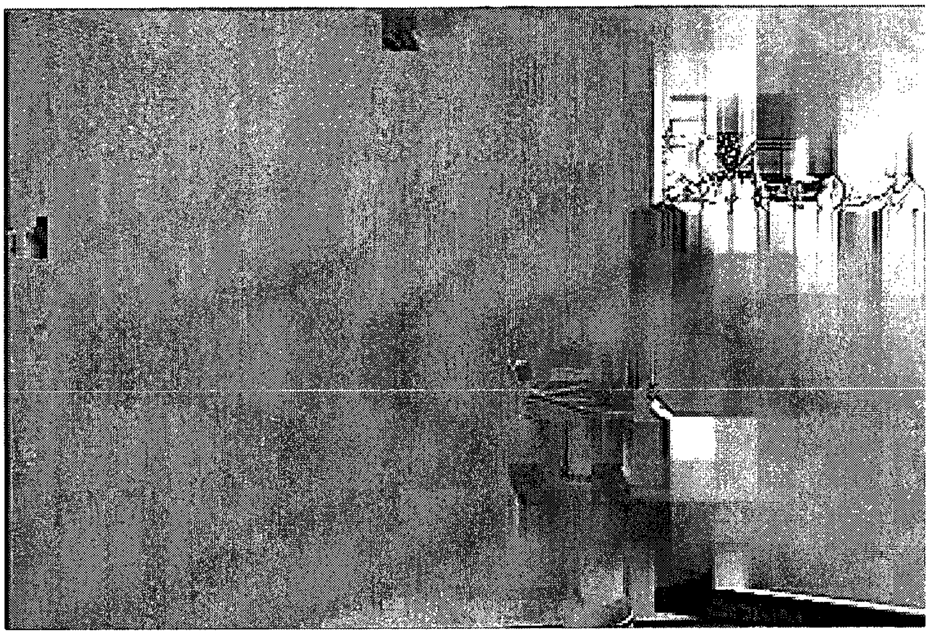


Figure 3.46: Frame 100 of video sequence “Table-Tennis” transmitted over an AWGN channel at 1.7dB channel SNR using convolutional decoding only.

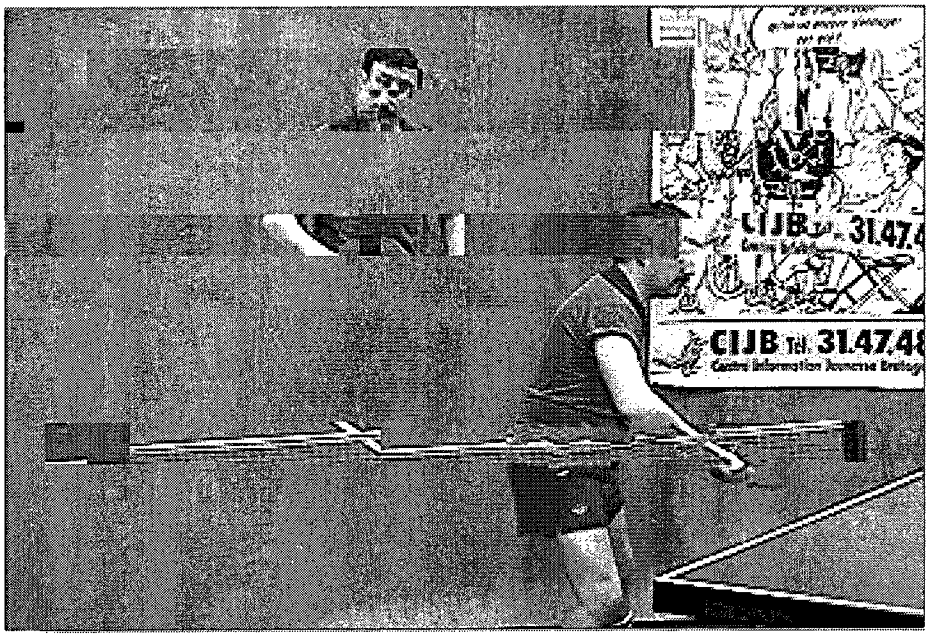


Figure 3.47: Frame 100 of video sequence “Table-Tennis” decoded by the IJSCD-I scheme at channel SNR of 1.7dB and at the 4th iteration.

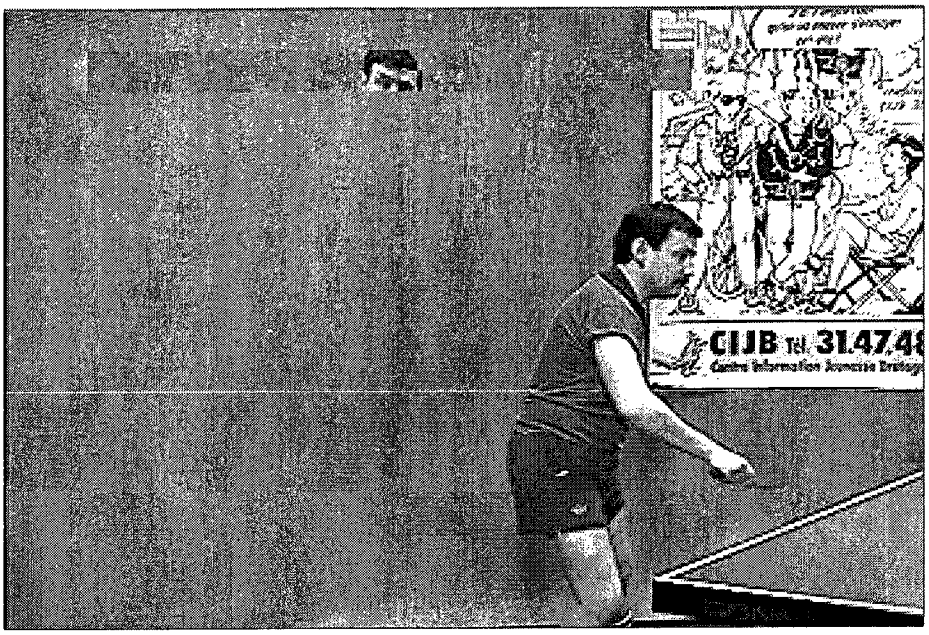


Figure 3.48: Frame 100 of video sequence “Table-Tennis” decoded by the IJSCD-VC-PI scheme at channel SNR of 1.7dB and at the 4th iteration.

3.9 Summary

This chapter presented the IJSCD-VC scheme in which both the redundancy systematically added by the channel coding and the H.264 source semantic residual redundancy in the compressed video are incorporated, in an iterative manner, to detect and correct transmission errors. A new checking method, namely VC, was proposed. VC uses information from slice candidates that failed semantic verification to check the semantic correctness of the current slice candidate without running the verification. Simulation results show that, using VC can reduce the computational complexity while keeping the same performance (IJSCD-VC-FS scheme). Alternatively, IJSCD-VC-PI scheme achieves a performance improvement over the IJSCD-I scheme proposed in [5] while keeping almost the same computational complexity.

Chapter 4
Iterative Joint Source-Channel Decoding Using Voting and Virtual Checking Method (IJSCD-VVC)

Chapter 3 presented the IJSCD-VC scheme. A new checking method, namely VC, was proposed, which uses information from slice candidates that failed semantic verification to virtually check the current slice candidate. The Modifier used in that scheme was proposed in [5], in which soft values of bits are modified according to their hard values in the BSC.

Chapter 4 proposes another IJSCD scheme, called Iterative Joint Source-Channel Decoding using Voting and Virtual Checking method (IJSCD-VVC). Like the last chapter the work in this chapter will use slice candidates that failed semantic verification but it will use them in the Modifier. Specifically, a new modification scheme is proposed to modify soft values of bits before feeding them back into the channel decoder for the next iteration by taking into account whether the BSC passes the verification or not. It is believed that besides the BSC, slice candidates that failed semantic verification also provide information about the correctness of bits. This new scheme is called the Voting method.

Chapter 4 is organized as follows: An experiment examining the delay in detecting errors is discussed in Section 4.1. Section 4.2 introduces the proposed IJSCD-VVC scheme for H.264 video transmission. The Modifier module is discussed in Section 4.3.

The performance as well as the computational complexity of the proposed scheme is evaluated in Section 4.4. The chapter concludes with a summary in Section 4.5.

4.1 Estimation of the Error Detection Delay in H.264

The modification method of soft values of bits proposed in this chapter takes into account how each of the slice candidates that didn't pass verification failed. Therefore, in this section, an experiment is performed to shed light on the question of when a bit error occurs in a video slice, how long it takes to be detected. Due to the nature of the compressed video, the location where a semantic error is detected by the SSV (i.e. the DB) is not the same as the location where it actually occurs. The distance in bits between the bit error location and the DB is called the Error Detection Delay (EDD). This section repeats the experiment performed in [41] to estimate the probability density function of EDD.

The experiment consists of inserting one bit error into a compressed video sequence for each run and many runs are performed for every bit in the data stream of the video sequence. In each run, if the source decoder can detect a semantic error, the value of the observed EDD is recorded. At the end of the experiment, the histogram of EDD as well as the CDF of EDD is plotted.

The experiment is performed with the two video sequences "Football" and "Table-Tennis". Each sequence is 4-second in length and has a frame size of 352x240 pixels. They are compressed using Main profile, CABAC entropy coding with a GOP length of 10 and GOP structure is IPPPP.

4.1.1 Simulation Results for Video "Football"

Figure 4.1 shows the histogram of EDD for video "Football". It is observed that after reaching a peak (within the first 20 bits), the curve drops down quickly in the range of [300, 500] and then slowly decreases to zero.

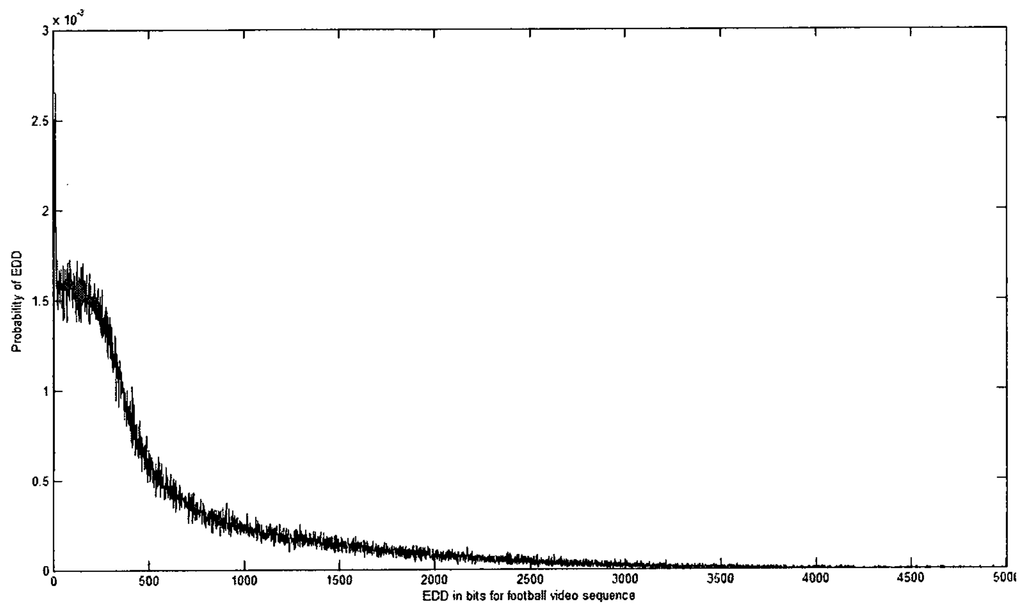


Figure 4.1: The histogram of EDD for video sequence "Football".

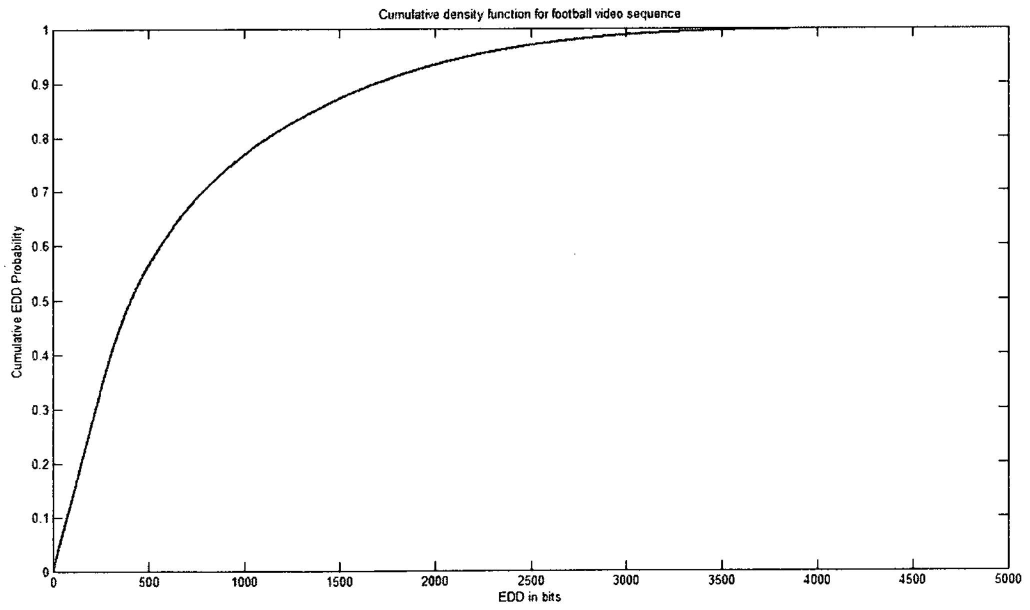


Figure 4.2: The CDF of EDD for video sequence "Football".

Figure 4.2 shows the CDF of EDD. The curve monotonically increases and has a knee point at around 1600 bits.

4.1.2 Simulation Results for Video "Table-Tennis"

Figure 4.3 shows the histogram of EDD for video "Table-Tennis". It has the same trends as what was observed for video "Football". Figure 4.4 shows the CDF of EDD for both videos "Table-Tennis" and "Football". It is observed that the CDF of EDD for "Table-Tennis" reaches the knee point faster than the CDF of EDD for "Football". This implies that, it takes more time on average to detect errors in video "Football" than in video "Table-Tennis".

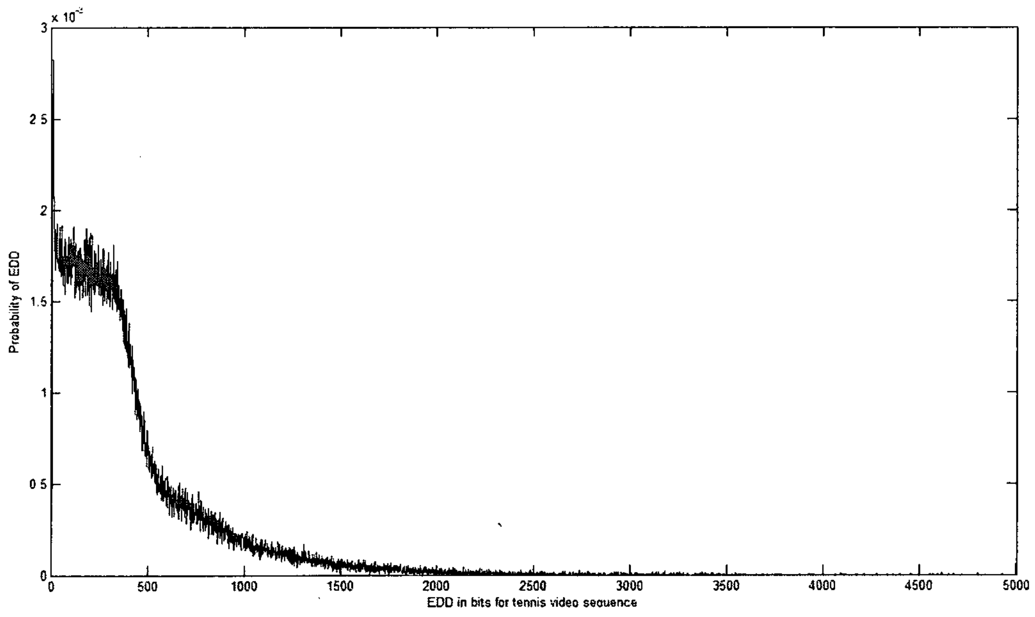


Figure 4.3: The histogram of EDD for video sequence "Table-Tennis".

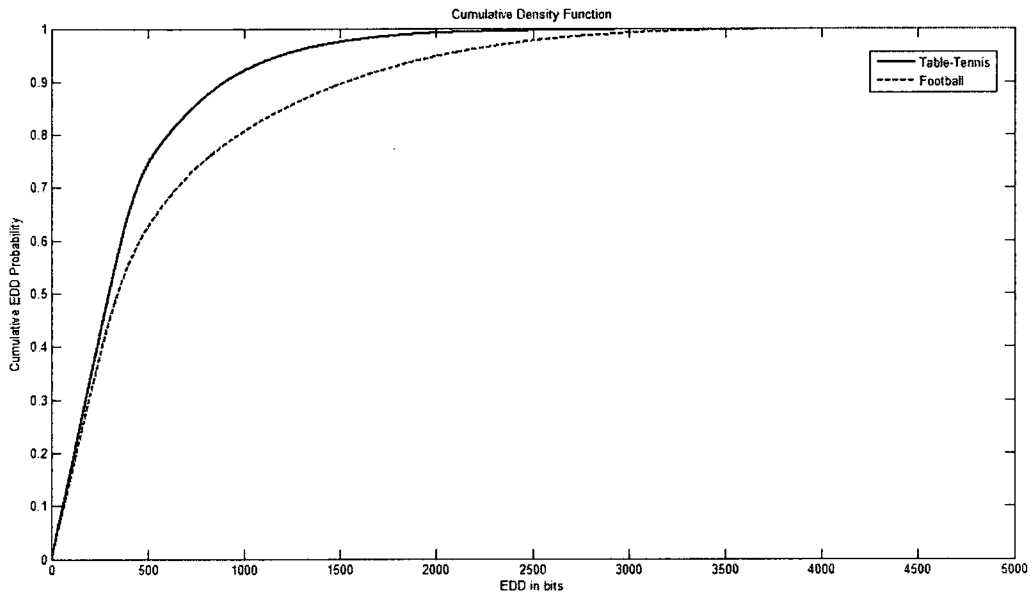


Figure 4.4: The CDF of EDD for video sequences "Table-Tennis" and "Football".

4.1.3 Implications of Simulation Results

Observing Figure 4.1 and 4.3, one can see that the curves showing the histogram of EDD have long tails. This means that the distance (in bits) between the DB and the true location of the bit error can vary greatly, from a short distance (within 5 bits for example) to a very far distance (up to 2000 bits for example). Hence, it is difficult to locate the exact location of a bit error based on the location of the DB.

However, these curves also show that in case the bit in question is in error, the probability that it is detected sooner is larger than the probability that it is detected later. In other words, a bit error is more likely to be detected after short delays than after long delays. Observing the histogram of EDD for video sequence "Football" (Figure 4.1), for instance, one can see the probability that a bit error is detected after 20 bits is about $2.8*10^{-3}$, while the probability that it is detected after 500 bits is about $0.5*10^{-3}$. From this observation, a method to evaluate the correctness of flip-bits in slices at the source decoder is proposed, namely the Voting method, which is used in the IJSCD-VVC scheme.

4.2 Iterative Joint Source-Channel Decoding Using Voting and Virtual Checking (IJSCD-VVC)

The proposed scheme, IJSCD-VVC, incorporates the redundancy systematically added by the channel coding and the source semantic residual redundancy in the H.264 compressed video in an iterative manner to detect and correct transmission errors. The block diagram of the IJSCD-VVC scheme is the same as the block diagram of the IJSCD-VC scheme shown in Figure 3.1.

All the modules in the diagram except the proposed Modifier were discussed in Chapter 3. The proposed Modifier is described in the following section.

4.3 The Proposed Modifier

The Modifier proposed in [5] uses the information from the BSC only. For example, if the hard value of a bit in question is 1 in the BSC, the Modifier [5] modifies the soft value of this bit in such a way that its *a-posteriori* probability of being 1 is increased.

In many cases, several slice candidates that failed semantic verification contain information about the correctness of several bits in the slice. In light of this, a new Modifier is proposed in this chapter. The proposed Modifier alters soft values of information bits depending on whether or not the BSC passes semantic verification. Sections below represent in detail the operation of the Modifier for two cases: when the BSC passes semantic verification and when it fails semantic verification.

4.3.1 Modification When the BSC Passes Semantic Verification

The experiment performed in Section 3.4.1 in Chapter 3 implies that if the H.264 decompressor does not detect any semantic errors in a slice, this slice has a high probability (over 99%) of not having any bit errors. Thus when the BSC has passed semantic verification (i.e. the H.264 decompressor does not detect any semantic errors in the BSC), it is very likely that almost every bit in the BSC is correct. Given this guideline, the soft value of the information bit u_k (where k is the time index) is modified according to its hard value in the BSC as follows:

$$L'(u_k|y_{[j]}) = L(u_k|y_{[j]}) + t \qquad (4.1)$$

Here $L'(u_k|y_{[j]})$ is the modified version of $L(u_k|y_{[j]})$. Note that $L(u_k|y_{[j]})$ is the soft value of bit u_k obtained from the output of the channel decoder at the j^{th} iteration, defined by Equation 3.2. The value of t is determined empirically by taking into account the hard value of bit u_k in the BSC (see Table 4.1).

Table 4.1: The choice of t for the modification of soft values of bits in case the BSC passes semantic verification

Value of bit u_k in the BSC	t
1	10
0	-10

This operation increases soft values (or reliabilities) of bits corresponding to their hard values in the BSC. One should note that for the case the BSC passes semantic verification, the modification method proposed in [5] modifies soft values of flip-bits and non-flip-bits differently (i.e. soft values of non-flip-bits are reinforced while soft values of flip-bits are scaled down and reversed). Meanwhile, the modification method proposed in this chapter reinforces soft values of flip-bits and non-flip-bits equally. Another important difference is that the value of t is chosen to be larger. This makes the soft values of bits more reliable so that they are decoded correctly in the next iteration. However, the value of t can't be too large either to avoid numerical problems (i.e. soft values of some bits obtained at the channel decoder reach infinity after three or four iterations).

4.3.2 Modification When the BSC Fails Semantic Verification

4.3.2.1 The General Idea of the Voting Method

In case that all slice candidates have failed semantic verification, a method, called the Voting method, is proposed to modify soft values of information bits. Rather than using the information from the BSC only (i.e. the slice candidate that failed semantic verification with the latest DB), this method uses the semantic verification results of several slice candidates that failed semantic verification to evaluate the correctness of flip-bits in the slice. Specifically, each slice candidate gives a vote (either a vote 1 or a vote 0) to each flip-bit in the slice by using the "Voting Rule". In this way, the *a posteriori* probabilities of flip-bits can be estimated.

For example, slice candidate $\boldsymbol{s}_m$ is about to vote for bit u_k. As discussed in Section 4.1, the experiment showed that: a bit error is more likely to be detected after short delays than after long delays. In light of this, if the distance in bits between bit u_k and the DB of $\boldsymbol{s}_m$ is far enough, it is likely that the error detected in $\boldsymbol{s}_m$ is not caused by bit u_k, but by another bit being in error. Consequently, the hard value of bit u_k in $\boldsymbol{s}_m$ (either 0 or 1) is more likely to be correct. Assuming that the hard value of bit u_k in s_m is 1, then s_m gives bit u_k a vote 1.

In contrast, if the distance between bit u_k and the DB of s_m is short enough, it is likely that the error detected in s_m is caused by bit u_k. Consequently, the hard value of bit u_k in s_m is more likely to be wrong. Assuming that the hard value of bit u_k in s_m is 1, then s_m gives bit u_k a vote 0.

4.3.2.2 The Voting Rule

This section discusses the Voting Rule that each slice candidate in the list (say slice candidate $s_m, m = 0 \ldots N_{MAX}$) uses to vote for flip-bit u_k (wh ere k is the time index). Denote:

- ❖ $d_{k,0}$: the DB of the PSC (note that the PSC flips no bit).
- ❖ $d_{k,1}$: the DB of the slice candidate that flips bit u_k only.
- ❖ $\boldsymbol{u} = (u_1, u_2, \ldots, u_k, \ldots, u_N) \in [0,1]^N$: the uncorrupted input information bit sequence, which is defined in Section 3.2.6 in Chapter 3.
- ❖ $y_{[j]} = (y_{1,j}, y_{2,j}, \ldots, y_{N,j}, \ldots y_{2N-1,j}, y_{2N,j}) \in \Re^{2N}$: the received noisy bit sequence at the j^th iteration. Note that $y_{[0]}$ is the originally received noisy bit sequence. $y_{[j]}$ has *2N* elements since the channel code rate is ½.
- ❖ N_{MAX} : the maximum number of slice candidates tested for each slice
- ❖ N_k : the number of slice candidates in the current slice participating in voting for bit u_k. Note that $N_k \le N_{MAX}$

- ❖ $\psi_{k,m} \in [0,1], k = 1..N, m = 1..N_k$: a vote for bit u_k given by slice candidate s_m
- ❖ λ_k : the total number of votes $u_k = 1$

Define the two thresholds (including the upper threshold and the lower threshold) as follows:

$$d_{k,upper} = \max(d_{k,0}, d_{k,1}) \tag{4.2}$$

And

$$d_{k,lower} = \min(d_{k,0}, d_{k,1}) \tag{4.3}$$

$d_{k,upper}$ and $d_{k,lower}$ are the two proposed thresholds used to set up the Voting Rule for bit u_k. Obviously, these two thresholds are the DBs detected when bit u_k is either 0 or 1 in the PSC. Note that these two thresholds are different for different flip-bits.

After the two thresholds are set up for bit u_k, slice candidate s_m gives a vote (either a vote 1 or a vote 0) to bit u_k as follows.

Denote $u_{km} \in [0,1]$ the value of bit position k in s_m (note that because the PSC is slice candidate $m = 0$, u_{k0} is the value of bit position k in the PSC); $\bar{u}_{km} \in [0,1]$ the inversed value of bit position k in s_m (i.e. if $u_{km} = 0$ then $\bar{u}_{km} = 1$ and vice versa); DB_m the DB detected in s_m. The Voting Rule is described as follows:

- ❖ If $DB_m \geq d_{k,upper}$, this means that the DB of s_m is far away from the location of bit u_k. Hence, it is likely that u_{km} is correct. Consequently s_m gives a vote $u_k = u_{km}$.
- ❖ If $DB_m \leq d_{k,lower}$, this means that the DB of s_m is close to the location of bit u_k. Hence, it is likely that u_{km} is incorrect. Consequently s_m gives a vote $u_k = \bar{u}_{km}$.

From this explanation, Table 4.2 generalizes the Voting Rule as follows.

Table 4.2: The Voting Rule that slice candidate s_m uses to vote for flip-bit u_k

If	u_{km}	The vote that slice candidate s_m gives to bit position k ($\psi_{k,m}$)
$DB_m \geq d_{k,upper}$ (u_{km} is correct)	0	0
	1	1
$DB_m \leq d_{k,lower}$ (u_{km} is incorrect)	0	1
	1	0

The zone between the two thresholds $d_{k,upper}$ and $d_{k,lower}$ is called the *null zone*. One should note that if the DB of slice candidate s_m falls into this zone (this means that $d_{k,lower} < DB_m < d_{k,upper}$), s_m is not allowed to vote for bit u_k. This is because if DB_m falls into this zone, it is very likely that there is the interaction caused by other flip-bit(s) which are close to bit u_k. In general, this interaction makes the vote that s_m gives to bit u_k unreliable.

One should also note that, in case $d_{k,upper} = d_{k,lower}$, this means flipping u_k does not change the location where the DB is detected in the PSC and thus the correctness of u_k can not be evaluated by slice candidates that failed semantic verification. This happens when the location of u_k is after the DB of the PSC. In such the case, u_k is said to not satisfy the Voting Rule and thus u_k is not voted.

The example that follows illustrates how each slice candidate votes for each flip-bit.

Bit Erros				10213					10372				
Flip-bits	10090	10112	10139	10213					10372		10649	10666	
Hard values of flip-bits in the PSC	0	1	1	0					0		1	1	
PSC(Slice candidate 0)								10285					
Slice candidate 1								10285			10649		
Slice candidate 2				10213						10464			
Slice candidate 3				10213						10464	10649		
Slice candidate 4								10285				10666	
Slice candidate 5			10139				10259						
Slice candidate 6		10112			10223								
Slice candidate 7			10139				10259				10649		
Slice candidate 8		10112			10223						10649		
Slice candidate 9			10139	10213									10679
Slice candidate 10	10090					10255							
Slice candidate 11								10289	10372				

Figure 4.5: 12 slice candidates in video "Football" are semantically verified. Their flip-bits, their DBs as well as the locations of bit errors are highlighted in yellow, green and pink respectively.

In this example, a slice in the video "Football" is considered. This slice has two bit errors locating at 10213 and 10372 in the bit stream, which is highlighted in pink. 12 slice candidates are generated for this slice. There are totally 7 flip-bits which are highlighted in yellow (including 10090, 10112, 10139, 10213, 10372, 10649, 10666). Each slice candidate is semantically verified and its DB is highlighted in green. The voting process for some flip-bits is demonstrated as follows.

Vote for flip-bit 1 locating at 10090:

The DB of the PSC is 10285.

The DB of Slice candidate 10 (the one that flips flip-bit 1 only) is 10255.

Applying Equation 4.2 and 4.3, the two thresholds for flip-bit 1 are: $d_{1,upper} = 10285$ and $d_{1,lower} = 10225$. Table 4.3 demonstrates the voting result for flip-bit 1.

Table 4.3: The Voting table made for flip-bit 1 located at 10090

Slice candidate	$DB_m, (m = 0..11)$	**Original value of flip-bit 1 in each candidate**	**Votes for flip-bit 1** $\psi_{1,m}, (m = 0..11)$
PSC	10285	0	0
Slice candidate 1	10285	0	0
Slice candidate 2	10464	0	0
Slice candidate 3	10464	0	0
Slice candidate 4	10285	0	0
Slice candidate 5	10259	0	Not vote
Slice candidate 6	10223	0	1
Slice candidate 7	10259	0	Not vote
Slice candidate 8	10223	0	1
Slice candidate 9	10679	0	0
Slice candidate 10	10255	1	0
Slice candidate 11	10285	0	0

Note that Slice Candidates 5 and 7 have their DBs (at 10259) fall into the null zone (i.e. [10225, 10285]), thus they are not allowed to vote for flip-bit 1.

In summary, there are 10 slice candidates voting for flip-bit 1. This bit receives 8 votes 0 and 2 votes 1.

Vote for flip-bit 2 locating at 10112:

The DB of the PSC is 10285.

The DB of Slice candidate 6 (the one that flips flip-bit 2 only) is 10223.

Applying Equation 4.2 and 4.3, the two thresholds for flip-bit 2 are: $d_{2,upper} = 10285$ and $d_{2,lower} = 10223$. Table 4.4 demonstrates the voting result for flip-bit 2.

Table 4.4: The Voting table made for flip-bit 2 located at 10112

Slice candidate	$DB_m,(m=0..11)$	**Original value of flip-bit 2 in each candidate**	**Votes for flip-bit 2** $\psi_{2,m},(m=0..11)$
PSC	10285	1	1
Slice candidate 1	10285	1	1
Slice candidate 2	10464	1	1
Slice candidate 3	10464	1	1
Slice candidate 4	10285	1	1
Slice candidate 5	10259	1	Not vote
Slice candidate 6	10223	0	1
Slice candidate 7	10259	1	Not vote
Slice candidate 8	10223	0	1
Slice candidate 9	10679	1	1
Slice candidate 10	10255	1	Not vote
Slice candidate 11	10285	1	1

In summary, there are 9 slice candidates vote for flip-bit 2. This bit receives 9 votes 1.

Vote for flip-bit 4 locating at 10213:

The DB of the PSC is 10285.

The DB of Slice candidate 2 (the candidate that flips flip-bit 4 only) is 10464.

Applying Equation 4.2 and 4.3, the two thresholds for flip-bit 4 are: $d_{4,upper}=10464$ and $d_{4,lower}=10285$. Table 4.5 demonstrates the voting result for flip-bit 4.

Table 4.5: The Voting table made for flip-bit 4 located at 10213

Slice candidate	$DB_m,(m=0..11)$	**Original value of flip-bit 4 in each candidate**	**Votes for flip-bit 4** $\psi_{4,m},(m=0..11)$
PSC	10285	0	1
Slice candidate 1	10285	0	1
Slice candidate 2	10464	1	1
Slice candidate 3	10464	1	1
Slice candidate 4	10285	0	1
Slice candidate 5	10259	0	1
Slice candidate 6	10223	0	1
Slice candidate 7	10259	0	1
Slice candidate 8	10223	0	1
Slice candidate 9	10679	1	1
Slice candidate 10	10255	0	1
Slice candidate 11	10285	0	1

In summary, there are 12 slice candidates voting for flip-bit 4. It receives 12 votes 1.

Vote for flip-bits locating at 10372, 10649, 10666:

These three flip-bits are not voted by any slice candidates since they do not satisfy the Voting Rule.

4.3.2.3 The Source Intrinsic Information

After the voting process is completed, a new soft value, namely the *source intrinsic information*, can be estimated for each flip-bit that has experienced the voting process as follows.

The total number of votes 1 for bit u_k can be calculated as the sum of N_k votes consisting only of zeros and ones:

$$\lambda_k = \sum_{m=1}^{N_k} \psi_{k,m} \tag{4.4}$$

Denote $P(u_k = 1|\lambda_k)$ and $P(u_k = 0|\lambda_k)$ the *a posteriori* probabilities of bit u_k estimated at the Source Decoder. Specifically, $P(u_k = 1|\lambda_k)$ is the probability that bit u_k is 1 given that it receives N_k votes. Similarly, $P(u_k = 0|\lambda_k)$ is the probability that bit u_k is 0 given that it receives N_k votes. They can be derived as follows:

$$P(u_k = 1|\lambda_k) = \frac{\lambda_k}{N_k} \tag{4.5}$$

And

$$P(u_k = 0|\lambda_k) = 1 - \frac{\lambda_k}{N_k} \tag{4.6}$$

The source intrinsic information of bit u_k obtained at the Source Decoder, denoted $L(u_k|\lambda_k)$, is the *a posteriori* log-likelihood ratio defined as follows:

$$L(u_k|\lambda_k) = \ln\left(\frac{P(u_k = 1|\lambda_k)}{P(u_k = 0|\lambda_k)}\right) = \ln\left(\frac{\frac{\lambda_k}{N_k}}{1 - \frac{\lambda_k}{N_k}}\right) \tag{4.7}$$

Finally, the source intrinsic information of bit u_k is:

$$L(u_k|\lambda_k) = \ln\left(\frac{\lambda_k}{N_k - \lambda_k}\right) \tag{4.8}$$

Note that when $\lambda_k = N_k$ (this means that $P(u_k = 1|\lambda_k) = 1$) or $\lambda_k = 0$ (this means that $P(u_k = 1|\lambda_k) = 0$), the source intrinsic information of bit u_k reaches infinity or minus infinity, which causes numerical difficulties. In order to avoid this problem, $P(u_k = 1|\lambda_k)$ is set to be 0.999999 if $\lambda_k = N_k$. And $P(u_k = 1|\lambda_k)$ is set to be 0.000001 if $\lambda_k = 0$

4.3.2.4 Combine Channel Soft Values with Source Intrinsic Information

After the channel decoder, each information bit has a soft value defined by Equation 3.2. After the source decoder, for the case when the BSC fails verification, several flip-bits have new soft values defined by Equation 4.8.

Finally, the modified soft value of bit u_k at the jth iteration, denoted $L'(u_k|y_{[j]}, \lambda_k)$, is the *a posteriori* log-likelihood ratio defined as follows:

$$L'(u_k|y_{[j]}, \lambda_k) = \ln\left(\frac{P(u_k = 1|y_{[j]}, \lambda_k)}{P(u_k = 0|y_{[j]}, \lambda_k)}\right) \tag{4.9}$$

Here, $P(u_k = 1|y_{[j]}, \lambda_k)$ and $P(u_k = 0|y_{[j]}, \lambda_k)$ are the *a posteriori* probabilities of bit u_k that has experienced the channel decoding and the source decoding.

Using Bayes methodology, we can write:

$$P(u_k = 1|y_{[j]}, \lambda_k) = \frac{P(y_{[j]}, \lambda_k|u_k = 1)P(u_k = 1)}{P(y_{[j]}, \lambda_k)} \tag{4.10}$$

It is assumed that $y_{[j]}, \lambda_k$ are independent. Roughly speaking, this assumption is rather fair since λ_k mainly depends on the hard value of bit u_k and the relative distance between the location of u_k and the DBs of slice candidates voting for u_k. It is also assumed that information bits are equally likely. Thus we can write:

$$P(y_{[j]}, \lambda_k|u_k = 1) = P(y_{[j]}|u_k = 1)P(\lambda_k|u_k = 1) \tag{4.11}$$

Similarly,

$$P(u_k = 0 | y_{[j]}, \lambda_k) = \frac{P(y_{[j]}, \lambda_k | u_k = 0) P(u_k = 0)}{P(y_{[j]}, \lambda_k)} \tag{4.12}$$

And

$$P(y_{[j]}, \lambda_k | u_k = 0) = P(y_{[j]} | u_k = 0) P(\lambda_k | u_k = 0) \tag{4.13}$$

Combining (4.10), (4.11), (4.12) and (4.13), we have:

$$L'(u_k | y_{[j]}, \lambda_k) = \ln\left(\frac{P(y_{[j]} | u_k = 1) P(\lambda_k | u_k = 1)}{P(y_{[j]} | u_k = 0) P(\lambda_k | u_k = 0)} \right) \tag{4.14}$$

The Bayes rule is used once again:

$$P(y_{[j]} | u_k = 1) = \frac{P(u_k = 1 | y_{[j]}) P(y_{[j]})}{P(u_k = 1)} \tag{4.15}$$

$$P(\lambda_k | u_k = 1) = \frac{P(u_k = 1 | \lambda_k) P(\lambda_k)}{P(u_k = 1)} \tag{4.16}$$

$$P(y_{[j]} | u_k = 0) = \frac{P(u_k = 0 | y_{[j]}) P(y_{[j]})}{P(u_k = 0)} \tag{4.17}$$

$$P(\lambda_k | u_k = 0) = \frac{P(u_k = 0 | \lambda_k) P(\lambda_k)}{P(u_k = 0)} \tag{4.18}$$

With the assumption that information bits are equally likely, we have $P(u_k = 0) = P(u_k = 1) = 1/2$. Hence, $L'(u_k | y_{[j]}, \lambda_k)$ can be further derived as follows:

$$L'(u_k | y_{[j]}, \lambda_k) = \ln\left(\frac{P(u_k = 1 | y_{[j]}) P(u_k = 1 | \lambda_k)}{P(u_k = 0 | y_{[j]}) P(u_k = 0 | \lambda_k)} \right) = \ln\left(\frac{P(u_k = 1 | y_{[j]})}{P(u_k = 0 | y_{[j]})} \right) + \ln\left(\frac{P(u_k = 1 | \lambda_k)}{P(u_k = 0 | \lambda_k)} \right) \tag{4.19}$$

Finally, the modified soft value of bit u_k at the jth iteration for the case when the BSC fails verification is:

$$L'(u_k \mid y_{[j]}, \lambda_k) = L(u_k \mid y_{[j]}) + L(u_k \mid \lambda_k) \quad (4.20)$$

Here $L(u_k \mid y_{[j]})$ is the soft value of bit u_k obtained from the output of the channel decoder at the jth iteration defined by Equation 3.2. $L(u_k \mid \lambda_k)$ is the source intrinsic information of u_k defined by Equation 4.8. Equation 4.20 implies that, the modified soft value of bit u_k for the case the BSC fails verification is the sum of the soft value of u_k obtained at the channel decoder and the source intrinsic information of u_k obtained at the source decoder.

The modified soft valued bit stream is interleaved. This creates a new information bit stream, which is fed back into the channel decoder for the (j+1)th iteration as in [5].

4.3.2.5 The Choice of Sample Size

Sections 4.3.2.1 – 4.3.2.4 have discussed the Voting method. The advantage of the Voting method is that the correctness of the bit in question is evaluated by several slice candidates that experienced the semantic verification, not by the BSC only. However, not all slice candidates in the list vote for each flip-bit since the vote is only valid if it satisfies the Voting rule. Therefore, to make the voting result for each flip-bit reliable, the number of votes it receives must be large enough. From this explanation, denote N_t the minimum number of slice candidates voting for a bit required to make the voting result reliable. The value of N_t is estimated in this section.

Assume that votes which slice candidates give to bit u_k are independent. This leads to the fact that each vote is a Bernoulli trial and the voting process done for bit u_k is a

Bernoulli process. Consequently λ_k is a Binomial random variable [42]. Given λ_k is available, the probability that bit u_k is 1, denoted $P(u_k = 1|\lambda_k)$, can be estimated. In [42], the reliability of the estimation of $P(u_k = 1|\lambda_k)$ can be evaluated. In particular, it can be at least $(1-\mu).100\%$ confident that the error in estimating $P(u_k = 1|\lambda_k)$ will not exceed a specified amount e when the minimum number of slice candidates voting for bit u_k is [42]:

$$N_t = \frac{z_{\mu/2}^2}{4e^2} \tag{4.21}$$

Here, $z_{\mu/2}$ is the value of the standard normal curve above which an area of $\mu/2$ is found.

In this thesis, e is chosen to be 0.1 and μ is chosen to be 0.95. Hence, N_t can be estimated as follows:

$$N_t = \frac{z_{\mu/2}^2}{4e^2} = \frac{1.96^2}{4*0.1^2} \approx 100 \tag{4.22}$$

An experiment is set up to investigate the effect of changing N_t. The IJSCD-VVC scheme is run with the input video "Table-Tennis" at channel SNR of 1.6dB. The number of slice candidates actually verified is 300. 2500 slice candidates are generated for each slice for VC. The value of N_t varies from 1 to 2500. At each value of N_t, the simulation is run 8 times and with the average being taken.

Figure 4.6 shows the performance of the IJSCD-VVC scheme with different values of N_t at the 4th iteration. It is observed that changing N_t in the range from 1 to 1100 does not make significant change in terms of PSNR. In other words, the performance of the

IJSCD-VVC scheme is not very sensitive to N_t in the range of [1, 1100]. However, when N_t is further increased, the PSNR of the output video starts to decrease. Specifically, the PSNR of the output video in case $N_t = 2500$ is about 27.5dB, which is almost 3dB smaller than the PSNR of the output video in case $N_t = 100$. The reduction in PSNR is therefore significant. This is because when N_t is set too high (i.e. higher than 48% of the number of slice candidates generated for each slice for VC), this means that the voting result for a bit is accepted only if at least 48% in the total of 2500 slice candidates votes for it. This leads to the fact that the number of bits of which soft values are modified by the Voting method is reduced significantly. Consequently, this makes the performance of the scheme decrease.

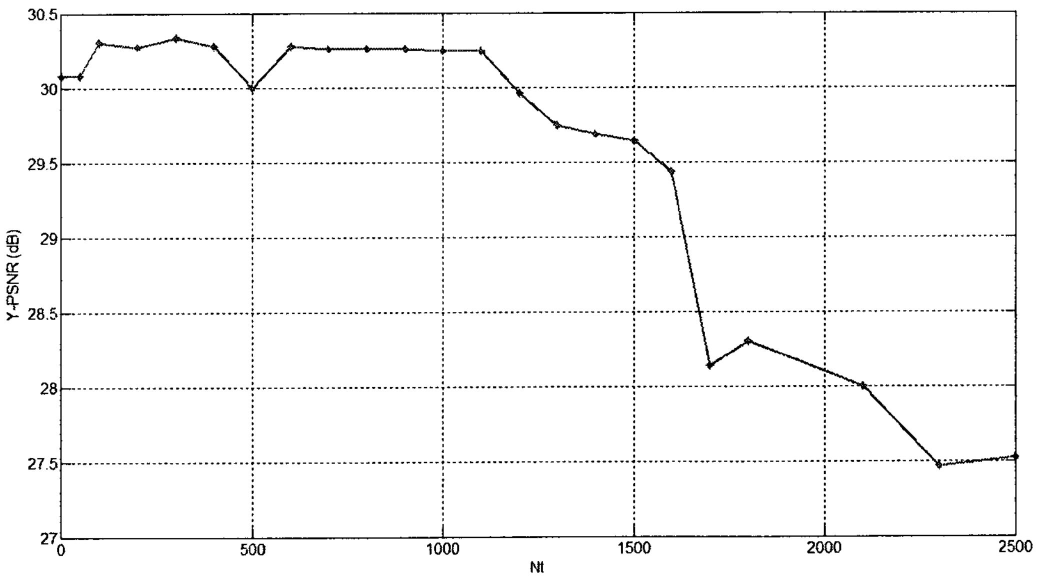


Figure 4.6: Compare the performance of the IJSCD-VVC scheme with different values of N_t : Y-PSNR vs. channel SNR for video "Table-Tennis" at channel SNR of 1.6dB and at the 4th iteration.

In summary, the voting result of a bit is only accepted if the number of slice candidates voting for it is equal or larger than N_t=100. In contrast, if the number of slice candidates voting for this bit is smaller than 100, the voting result is considered to be unreliable and thus it is discarded.

4.4 Simulation Results

The three video sequences "Football", "Table-Tennis" and "Garden" are used in the simulations. They are compressed using the same parameters that have been used in Chapter 3. In all of the experiments performed, the number of slice candidates actually verified is N_{SC}=300. The maximum number of slice candidates virtually checked is $N_{MAX} = 2500$. At each channel SNR value, simulations are run 15 times and with the average being taken. Also, in order to investigate the accuracy of the averaged values, the Chebyshev upper bound and lower bound are plotted with 90% of confidence.

4.4.1 Performance Evaluation

4.4.1.1 Simulation Results for Video "Football"

In this section, the performance of the proposed scheme IJSCD-VVC is evaluated using the video sequence "Football" as the input video.

4.4.1.1.1 Simulation Results for the Proposed Scheme IJSCD-VVC

Figure 4.7 and Figure 4.8 show the objective performance in terms of PSNR and BER for the IJSCD-VVC scheme. The scheme is performed up to the fourth iteration since there's almost no improvement after 4 iterations. Figure 4.9 shows the Chebyshev upper bound and lower bound with 90% of confidence for the Y-PSNR vs. channel SNR curve at the 4th iteration.

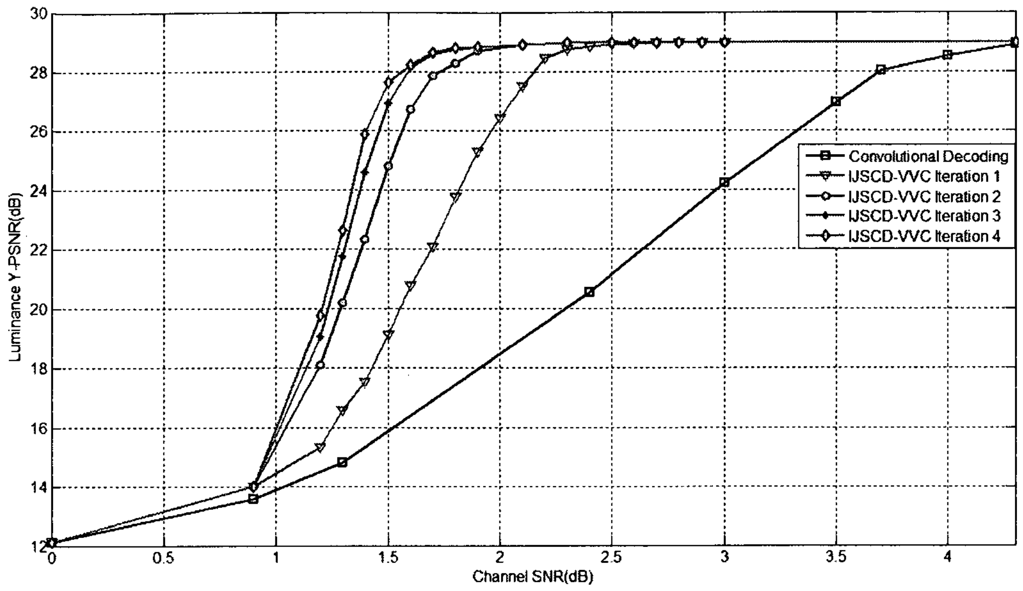


Figure 4.7: Y-PSNR vs. channel SNR for video "Football" of the IJSCD-VVC scheme.

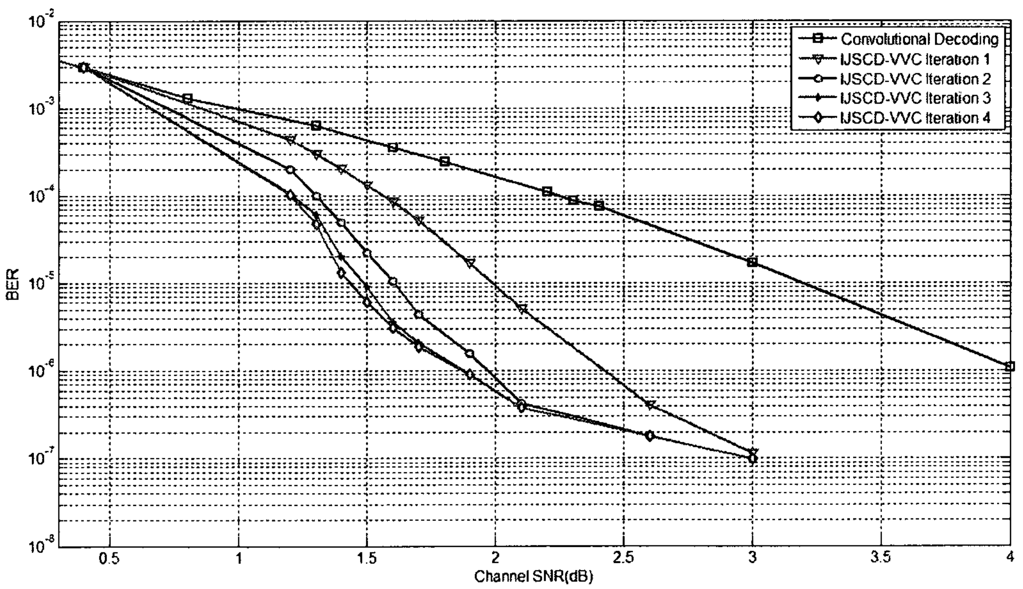


Figure 4.8: BER vs. channel SNR for video "Football" of the IJSCD-VVC scheme.

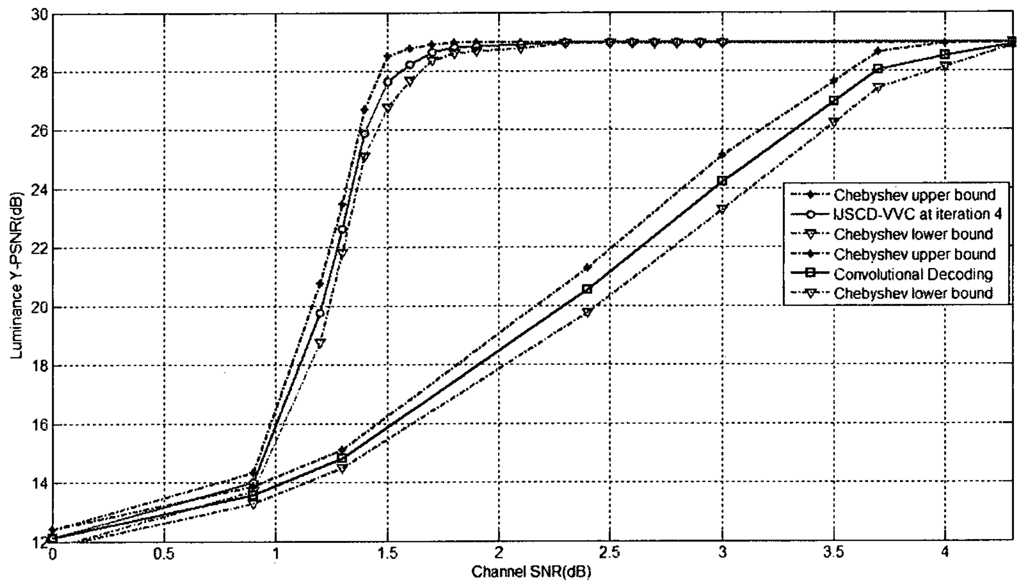


Figure 4.9: Chebyshev's upper bound and lower bound for Y-PSNR vs. channel SNR curves at the 4th iteration of the IJSCD-VVC scheme and at the output of the channel decoder for video "Football".

4.4.1.1.2 Compare Performance of Schemes at the 4th iteration

In this section, simulations are run to compare the performances of the following schemes at the 4th iteration:

- ❖ The scheme proposed by Levine *et.al.* in [5], which is called the IJSCD-I scheme.
- ❖ The IJSCD-VC-PI scheme proposed in Chapter 3.
- ❖ The proposed IJSCD-VVC scheme: in case the BSC passes verification, soft values of bits are modified using Equation 4.1. In case the BSC fails verification, soft values of bits are modified using the Voting Method.
- ❖ The IJSCD-VVC-II scheme: in case the BSC passes verification, soft values of bits are modified using Equation 3.24. In case the BSC fails verification, soft values of bits are modified using the Voting Method.

Figure 4.10 and Figure 4.11 compare the performance in terms of PSNR and BER of the four schemes at iteration 4th. One can see that, the maximum achievable PSNR is achieved at channel SNR of 2.2 dB when the IJSCD-I scheme is used. The IJSCD-VC-PI scheme achieves that value at channel SNR of 2.0dB. Meanwhile, the IJSCD-VVC scheme achieves that value at channel SNR of 1.8dB. With convolutional decoding, the maximum achievable PSNR is obtained with a channel SNR of 4.3dB or higher. In other words, without bandwidth expansion, the IJSCD-VVC scheme can achieve 0.4dB of channel SNR reduction over the IJSCD-I scheme, and up to 2.5dB of channel SNR reduction over convolutional decoding.

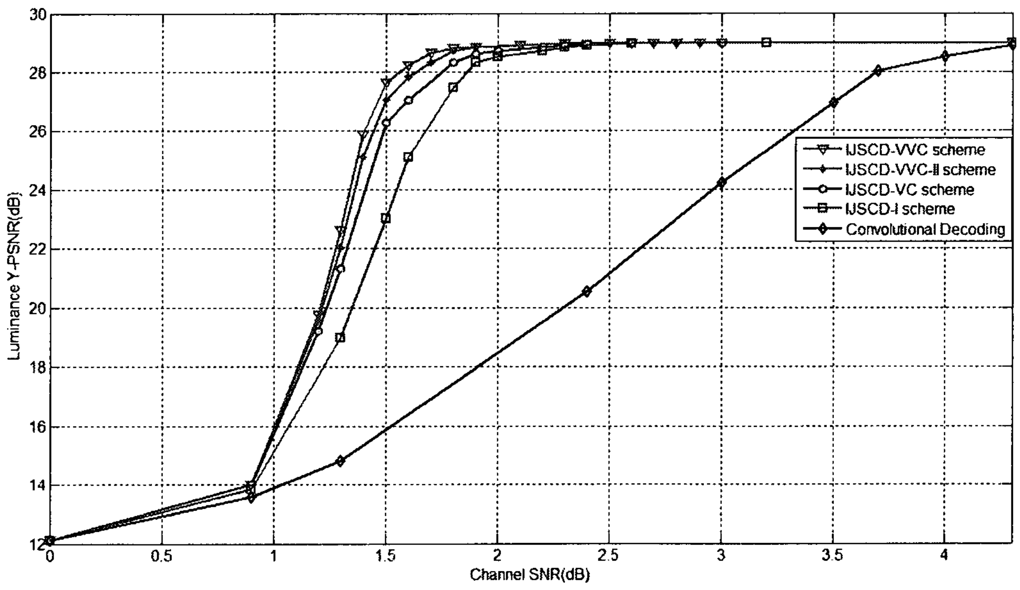


Figure 4.10: Comparison of IJSCD-I scheme, IJSCD-VC-PI scheme, IJSCD-VVC-II scheme and IJSCD-VVC scheme: PSNR vs. channel SNR obtained at the 4th iteration for video "Football".

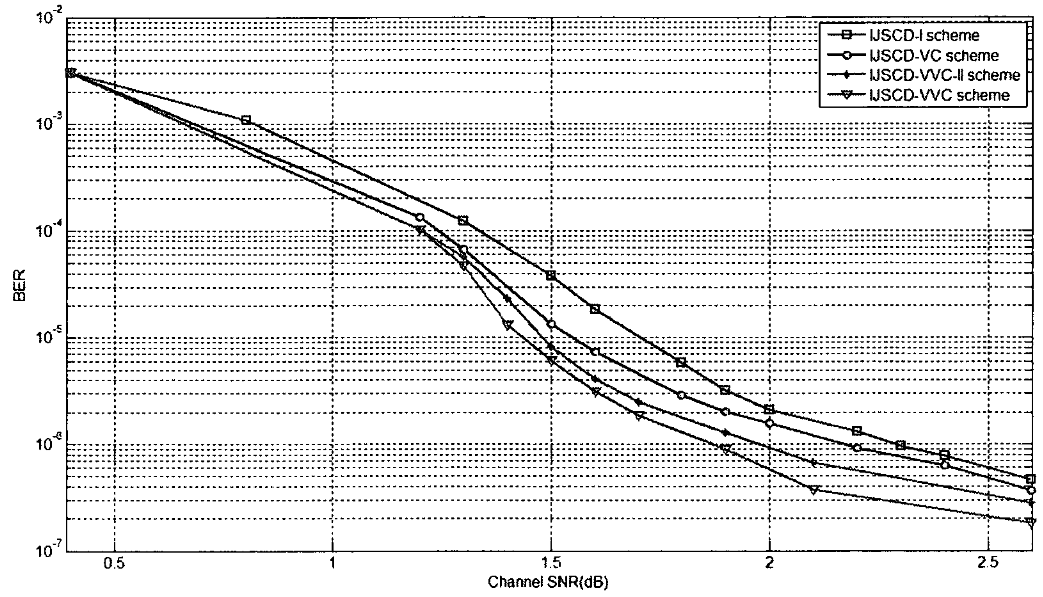


Figure 4.11: Comparison of IJSCD-I scheme, IJSCD-VC-PI scheme, IJSCD-VVC-II scheme and IJSCD-VVC scheme: BER vs. channel SNR obtained at the 4th iteration for video "Football".

4.4.1.1.3 Subjective Performance for Video "Football"

For the reader's own assessment, the resulting decompressed videos of the three schemes (IJSCD-I scheme, IJSCD-VC-PI scheme and IJSCD-VVC scheme) at the 4th iteration and of the convolutional decoding only are compared. In particular, frames 4 and 41 of the output videos of these schemes at channel SNR of 1.5dB are shown. For comparative purposes, the error-free frames are also presented. It is observed that the decompressed video using convolutional decoding only is not viewable since a large portion of each frame is damaged by black stripes and shifts. In contrast, the decompressed video using the IJSCD-I scheme obtained at the 4th iteration is viewable although there still remains some black stripes and some shifts. The qualit y of the decompressed video using the IJSCD-VC-PI scheme at the 4th iteration is further improved although there are still some visible errors. Meanwhile, the decompressed video using the IJSCD-VVC scheme obtained at the 4th iteration is the most viewable since almost all black stripes are removed and video slice shifts are corrected.

Figures below show decompressed results at frame 4.

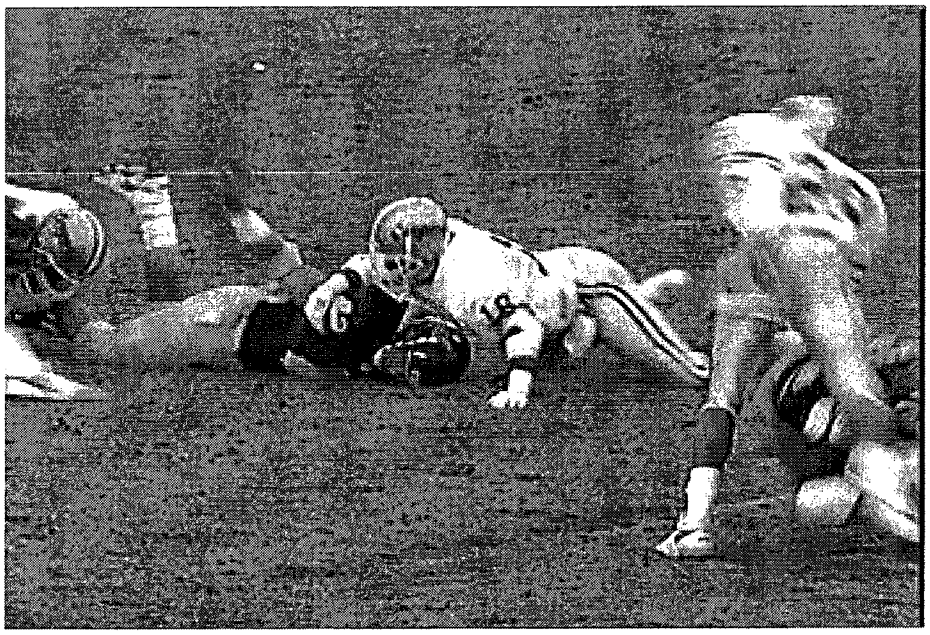


Figure 4.12: Decompressed error-free frame 4 of video sequence "Football".

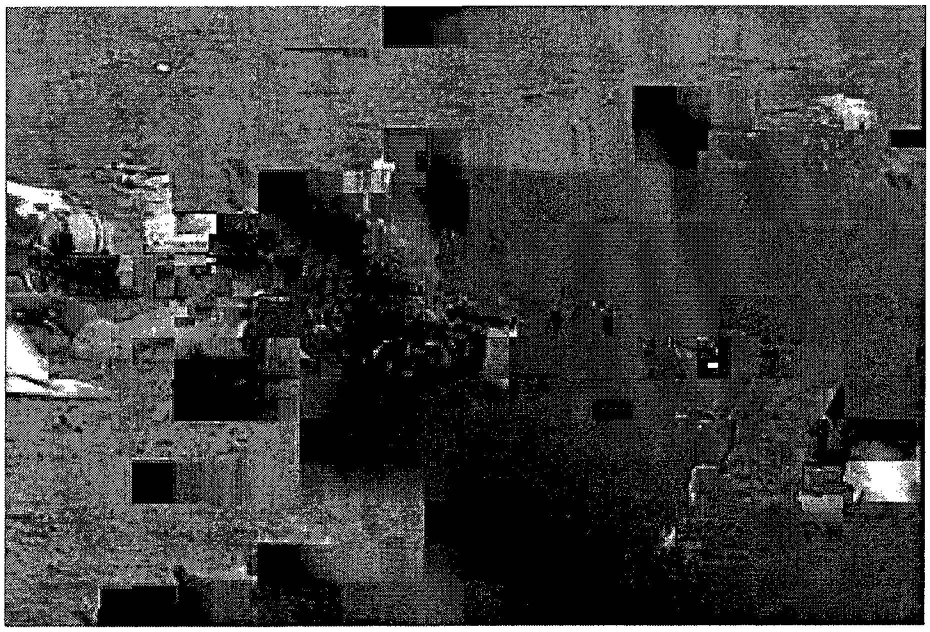


Figure 4.13: Frame 4 of video sequence "Football" decoded using convolutional decoding only at channel SNR of 1.5dB.

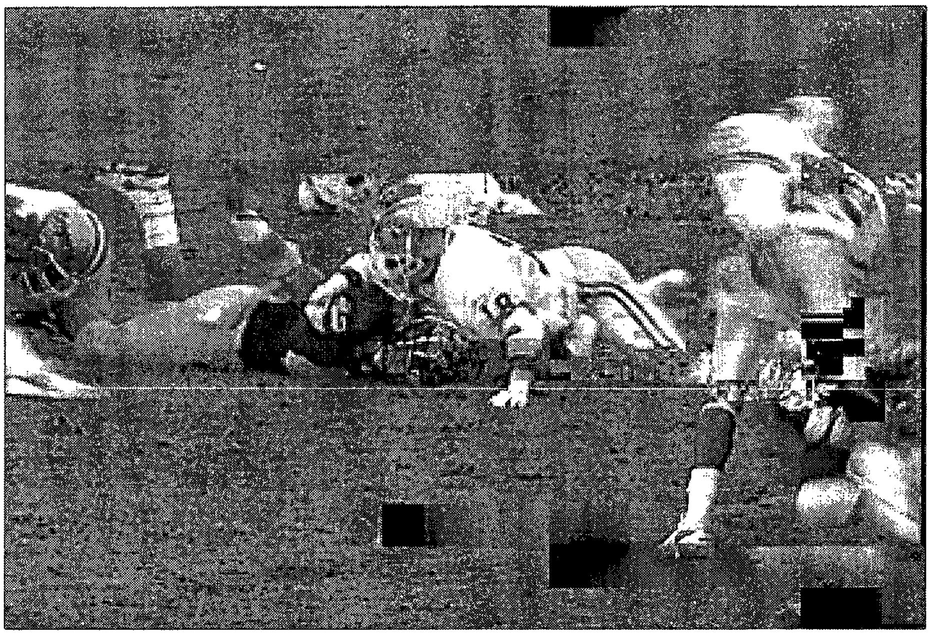


Figure 4.14: Frame 4 of video sequence "Football" decoded by the IJSCD-I scheme at channel SNR of 1.5dB and at the 4th iteration.

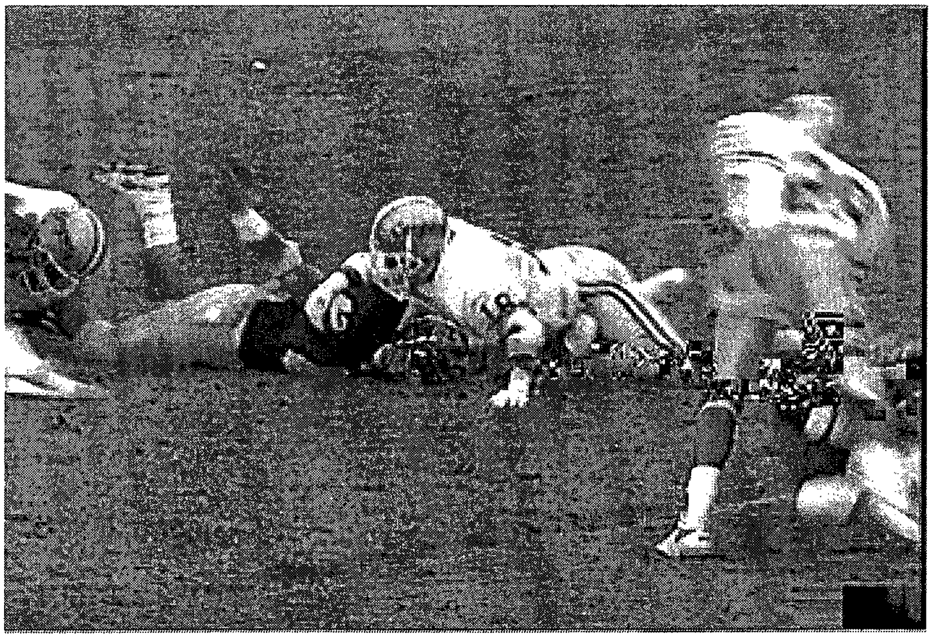


Figure 4.15: Frame 4 of video sequence "Football" decoded by the IJSCD-VC-PI scheme at channel SNR of 1.5dB and at the 4th iteration.

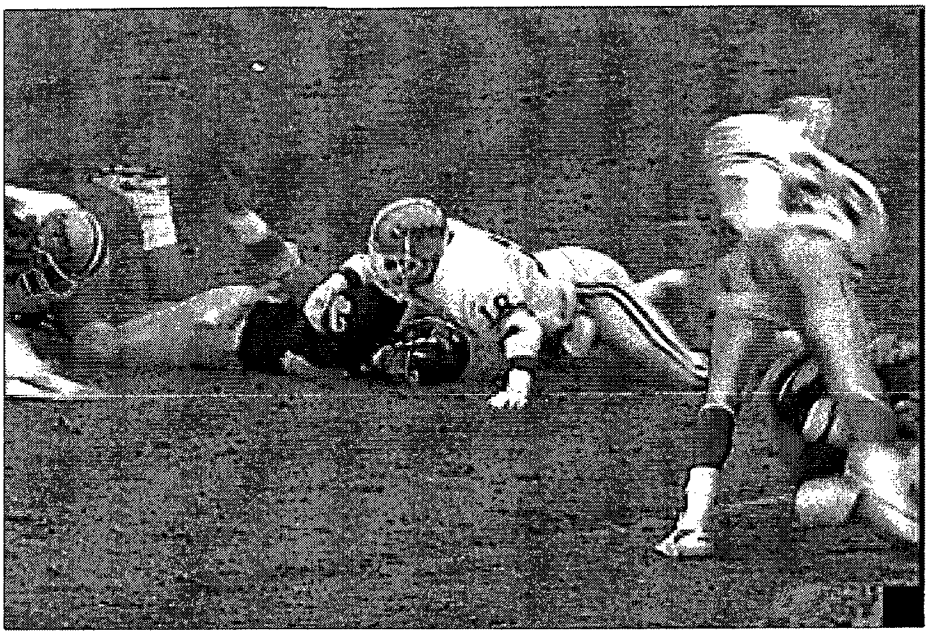


Figure 4.16: Frame 4 of video sequence "Football" decoded by the IJSCD-VVC scheme at channel SNR of 1.5dB and at the 4th iteration.

Figures below show decompressed results at frame 41.

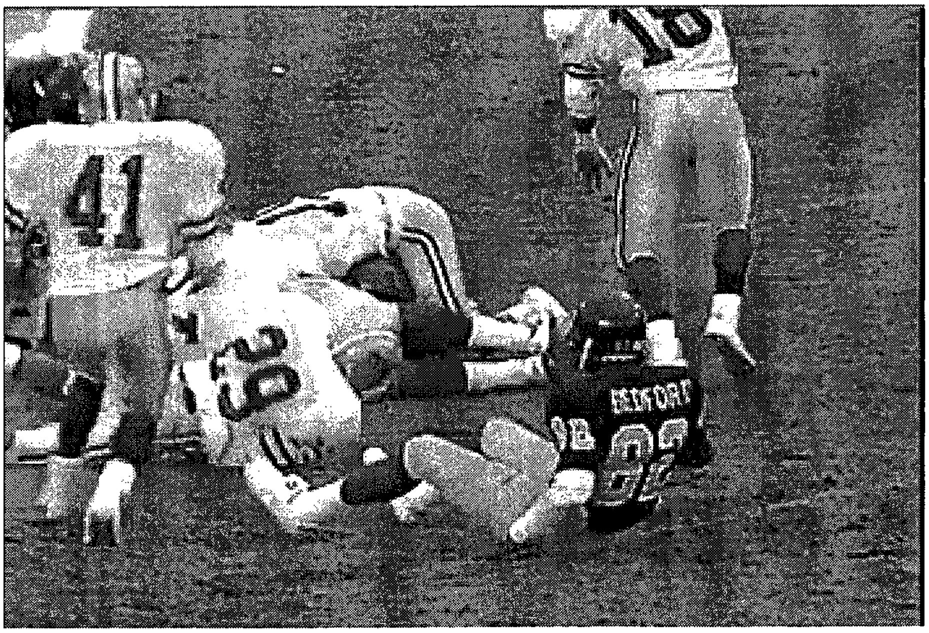


Figure 4.17: Decompressed error-free frame 41 of video sequence "Football".

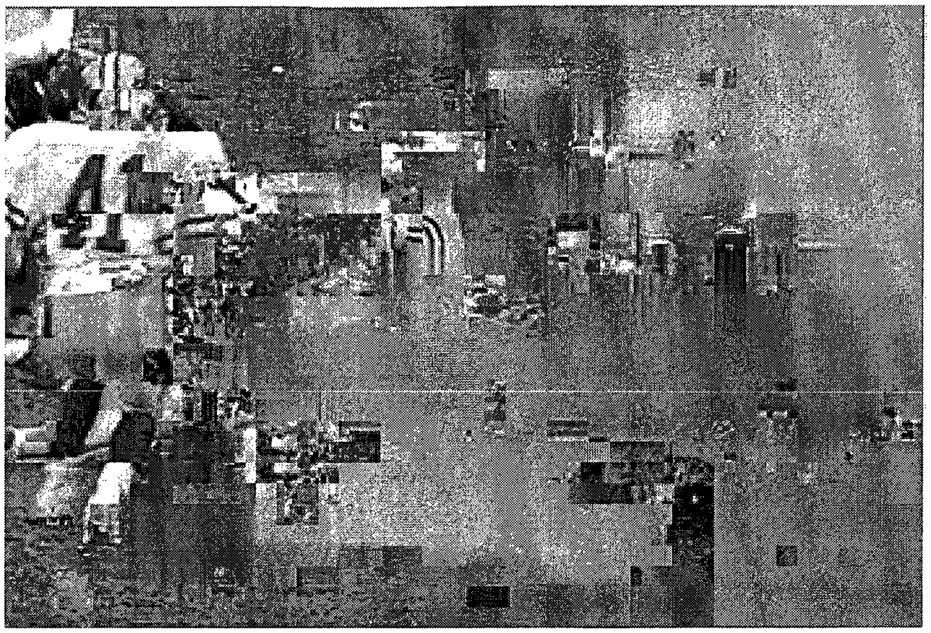


Figure 4.18: Frame 41 of video sequence "Football" decoded using convolutional decoding only at channel SNR of 1.5dB.

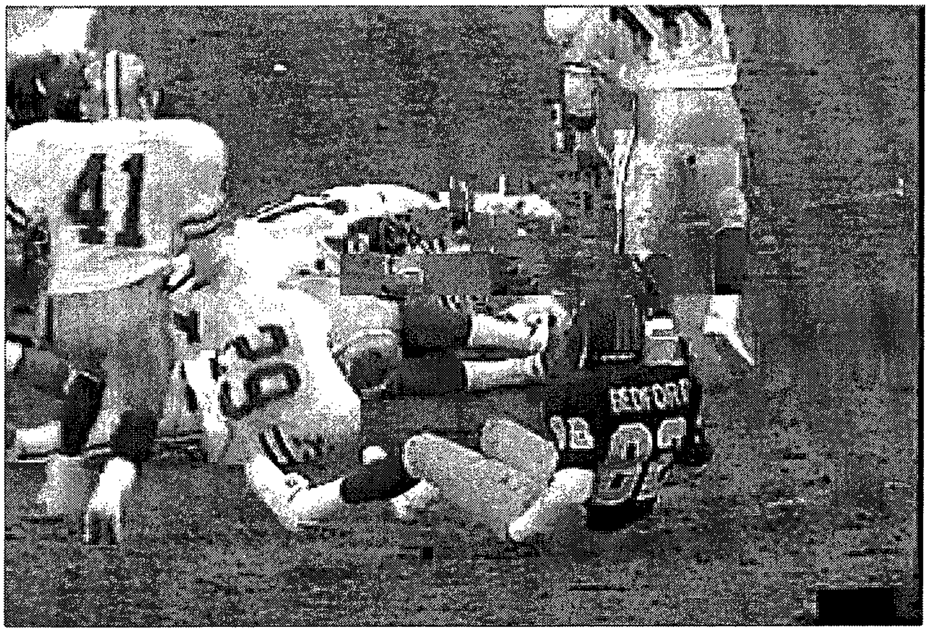


Figure 4.19: Frame 41 of video sequence "Football" decoded by the IJSCD-I scheme at channel SNR of 1.5dB and at the 4th iteration.

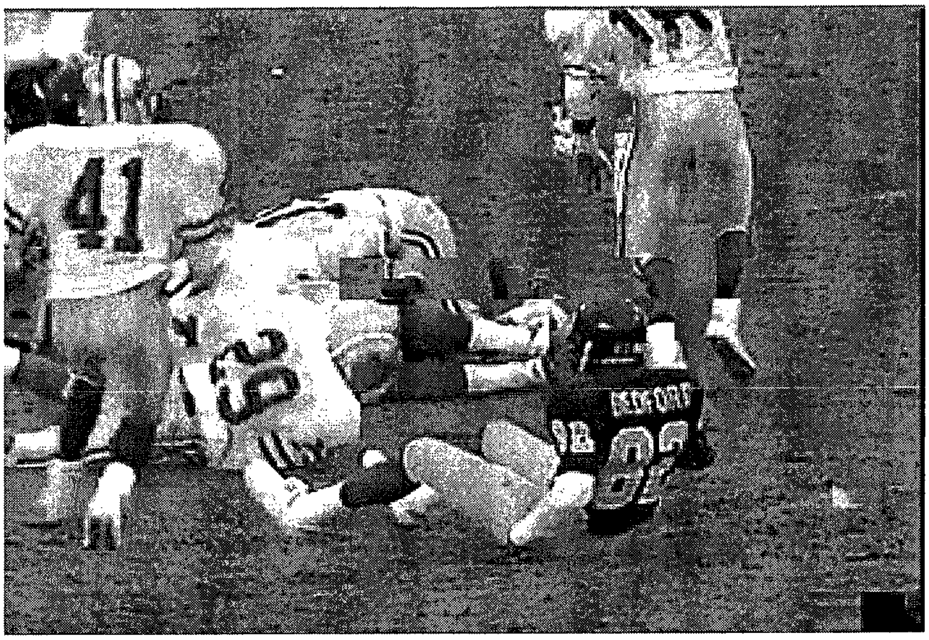


Figure 4.20: Frame 41 of video sequence "Football" decoded by the IJSCD-VC-PI scheme at channel SNR of 1.5dB and at the 4th iteration.

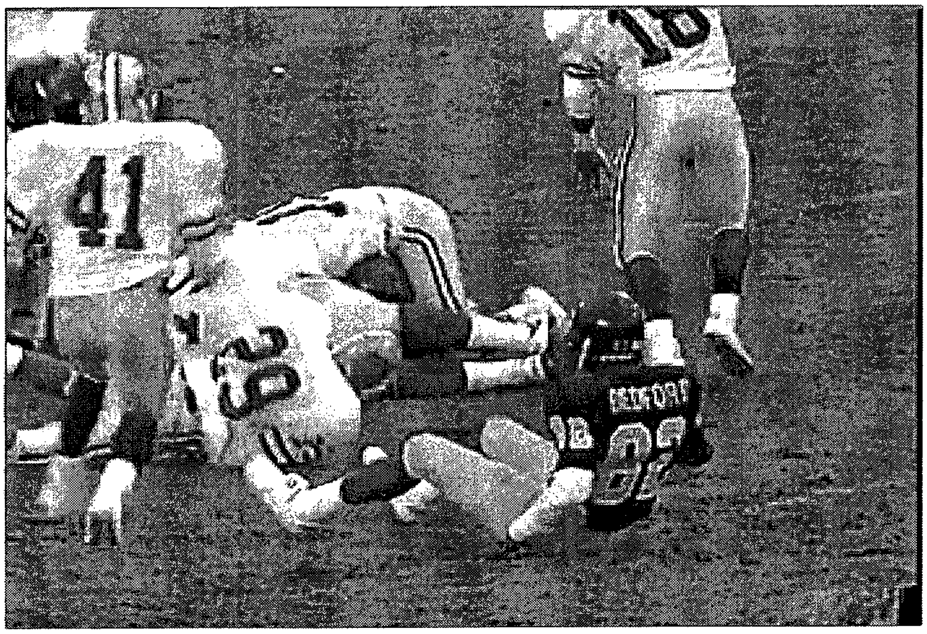


Figure 4.21: Frame 41 of video sequence "Football" decoded by the IJSCD-VVC scheme at channel SNR of 1.5dB and at the 4th iteration.

4.4.1.2 Simulation results for Video "Table-Tennis"

In this section, the performance of the proposed scheme IJSCD-VVC is evaluated using the video sequence "Table-Tennis" as the input video.

4.4.1.2.1 Simulation Results for the Proposed Scheme IJSCD-VVC

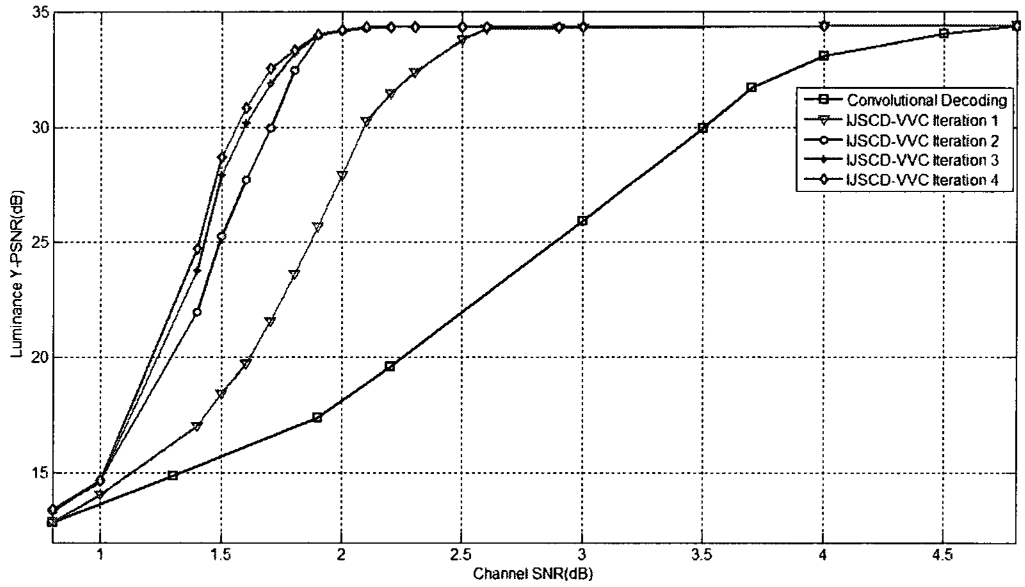


Figure 4.22: Y-PSNR vs. channel SNR for video "Table-Tennis" of the IJSCD-VVC scheme.

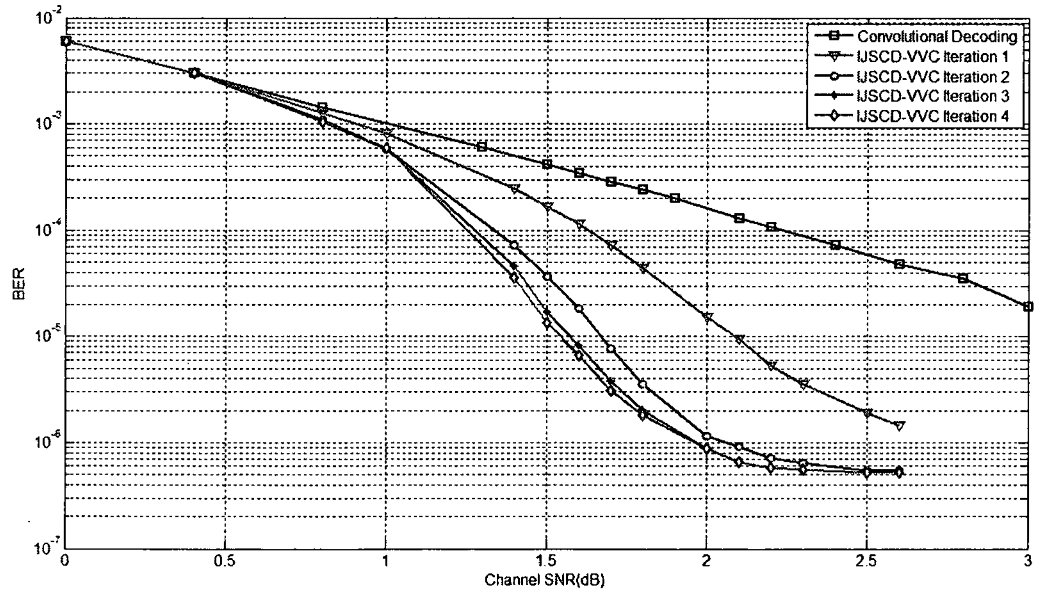


Figure 4.23: BER vs. channel SNR for video "Table-Tennis" of the IJSCD-VVC scheme.

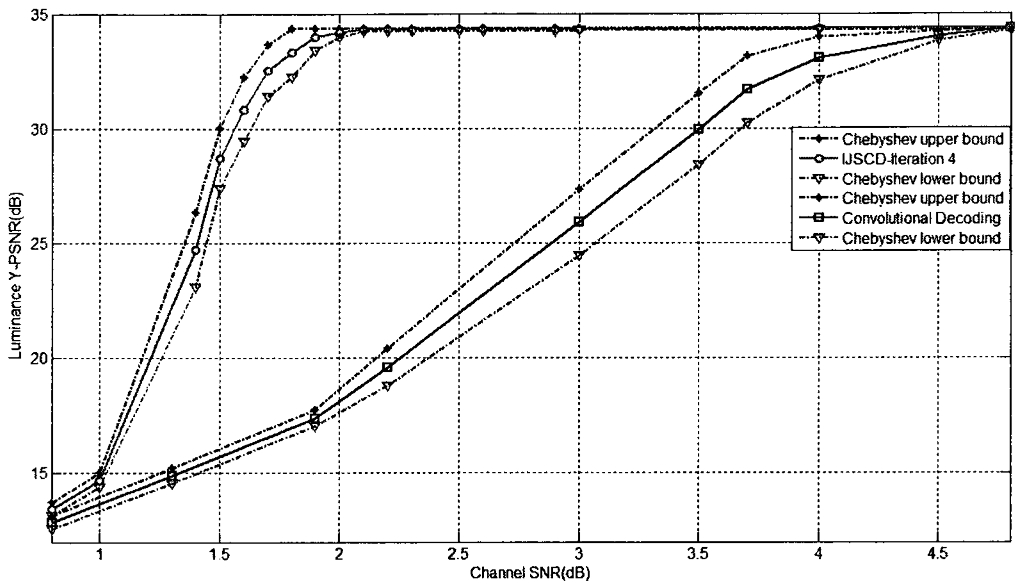


Figure 4.24: Chebyshev's upper bound and lower bound for Y-PSNR vs. channel SNR curves at the 4th iteration of the IJSCD-VVC scheme and at the output of the channel decoder for video "Table-Tennis".

Figure 4.22 and Figure 4.23 show the objective performance of the IJSCD-VVC scheme in terms of PSNR and BER for video "Table-Tennis" with 4 iterations. Figure 4.24 shows the Chebyshev upper bound and lower bound for the Y-PSNR vs. channel SNR curve of the scheme at the 4th iteration.

4.4.1.2.2 Compare Performance of Schemes at the 4th iteration

Figure 4.25 and Figure 4.26 compare the performance in terms of PSNR and BER of schemes at iteration 4th. One can see that, the maximum achievable PSNR is achieved at channel SNR of 2.5 dB when the IJSCD-I scheme is used. The IJSCD-VC-PI scheme achieves that value at channel SNR of 2.3dB. Meanwhile, the IJSCD-VVC scheme achieves that value at channel SNR of 2.0dB. With convolutional decoding, the maximum achievable PSNR is obtained with a channel SNR of 4.8dB or higher. In other words, without bandwidth expansion, the IJSCD-VVC scheme can achieve 0.5dB of channel SNR reduction over the IJSCD-I scheme, and up to 2.8dB of channel SNR reduction over convolutional decoding.

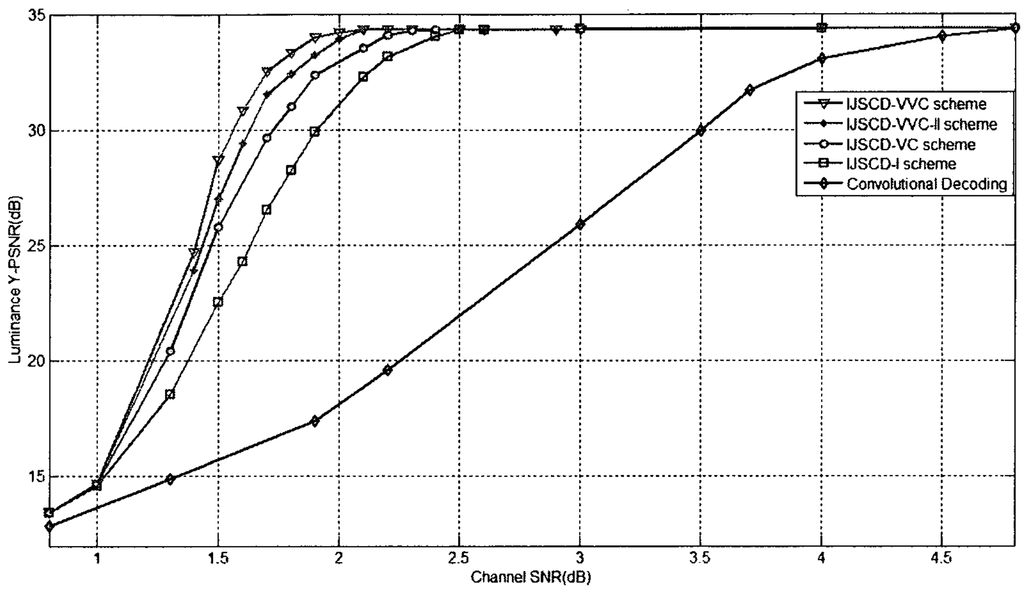


Figure 4.25: Comparison of IJSCD-I scheme, IJSCD-VC-PI scheme, IJSCD-VVC-II scheme and IJSCD-VVC scheme: PSNR vs. channel SNR obtained at the 4th iteration for video “Table-Tennis”.

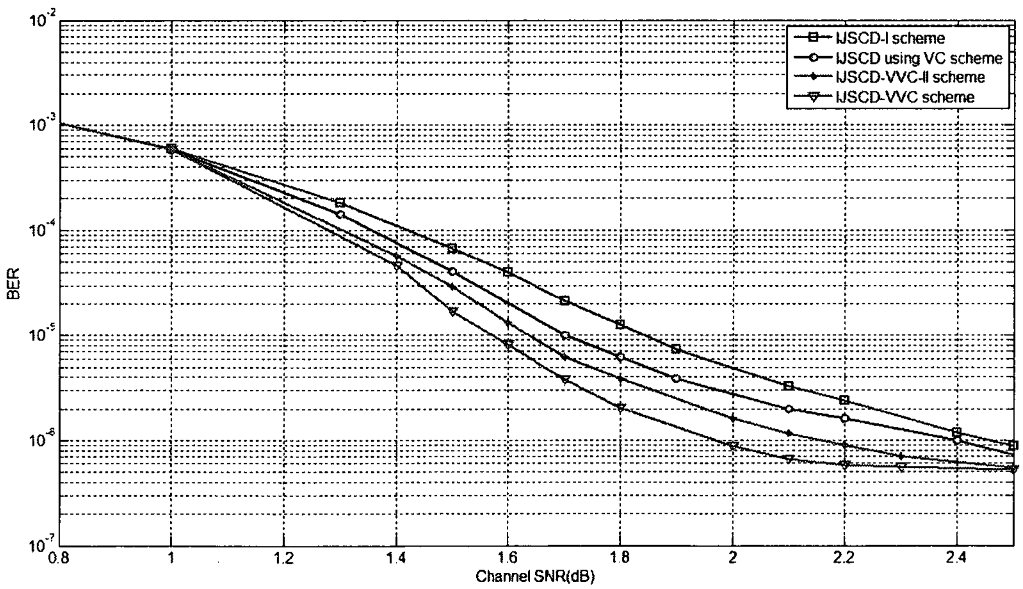


Figure 4.26: Comparison of IJSCD-I scheme, IJSCD-VC-PI scheme, IJSCD-VVC-II scheme and IJSCD-VVC scheme: BER vs. channel SNR obtained at the 4th iteration for video “Table-Tennis”.

4.4.1.2.3 Subjective Performance for Video "Table-Tennis"

For the reader's own assessment, the resulting decompressed videos of the three schemes (IJSCD-I scheme, IJSCD-VC-PI scheme and IJSCD-VVC scheme) at the 4th iteration and of the convolutional decoding only are compared. In particular, frames 4 and 42 of the output videos of these schemes at channel SNR of 1.5dB are shown. For comparative purposes, the error-free frames are also presented. As was observed for "Football", there are obvious improvements in picture quality when the IJSCD-VVC scheme is used in comparison to the convolutional decoding scheme, the IJSCD-I scheme and the IJSCD-VC-PI scheme.

Figures below show decompressed results at frame 4.

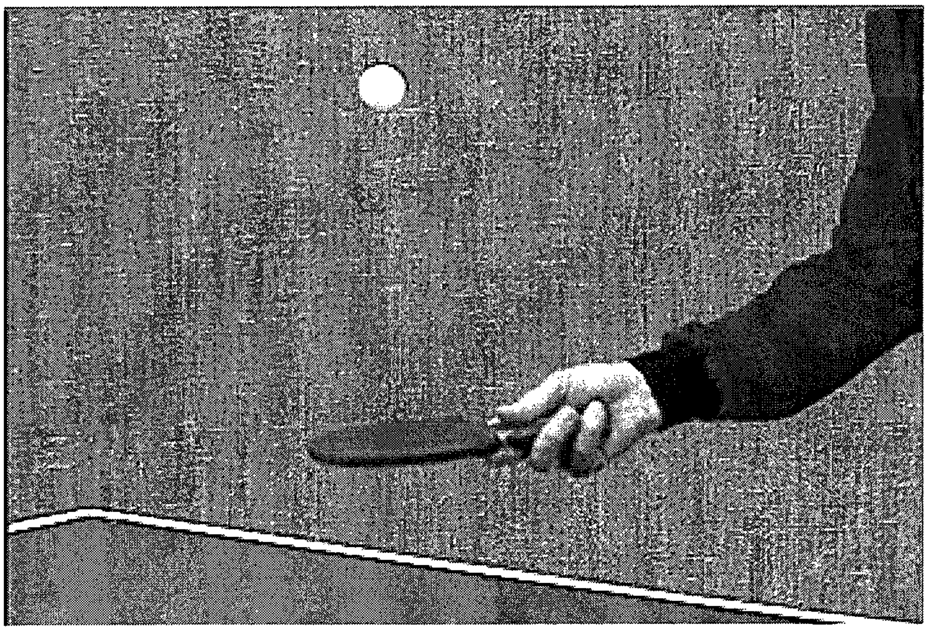


Figure 4.27: Decompressed error-free frame 4 of video sequence "Table-Tennis".

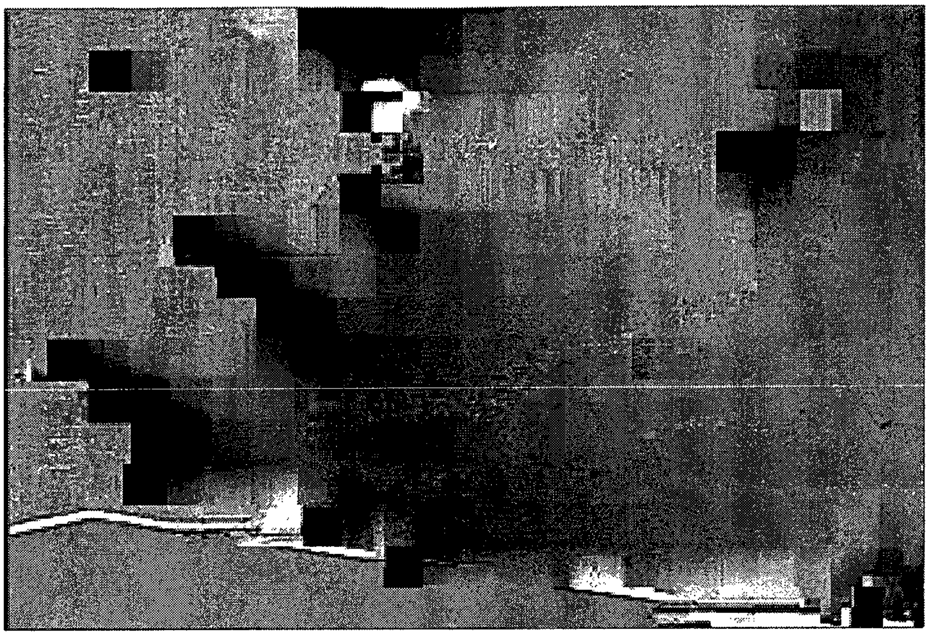


Figure 4.28: Frame 4 of video sequence "Table-Tennnis" decoded using convolutional decoding only at channel SNR of 1.5dB.

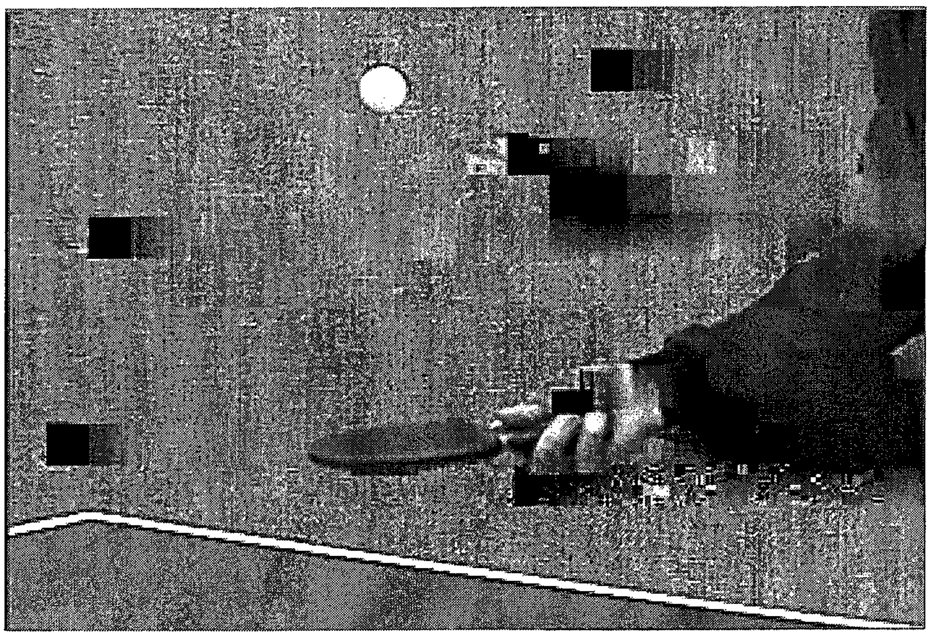


Figure 4.29: Frame 4 of video sequence "Table-Tennis" decoded by the IJSCD-I scheme at channel SNR of 1.5dB and at the 4th iteration.

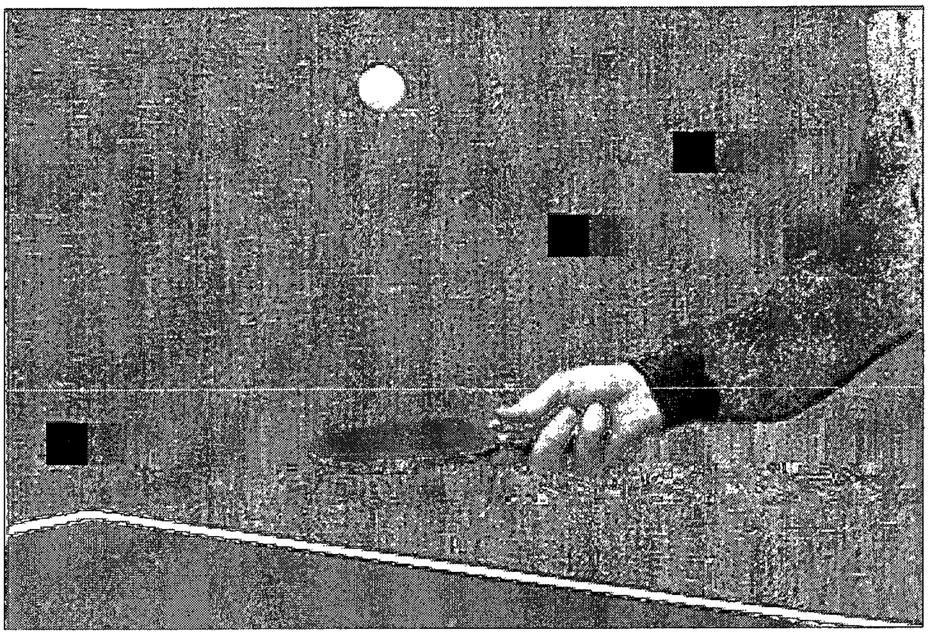


Figure 4.30: Frame 4 of video sequence "Table-Tennis" decoded by the IJSCD-VC-PI scheme at channel SNR of 1.5dB and at the 4th iteration.

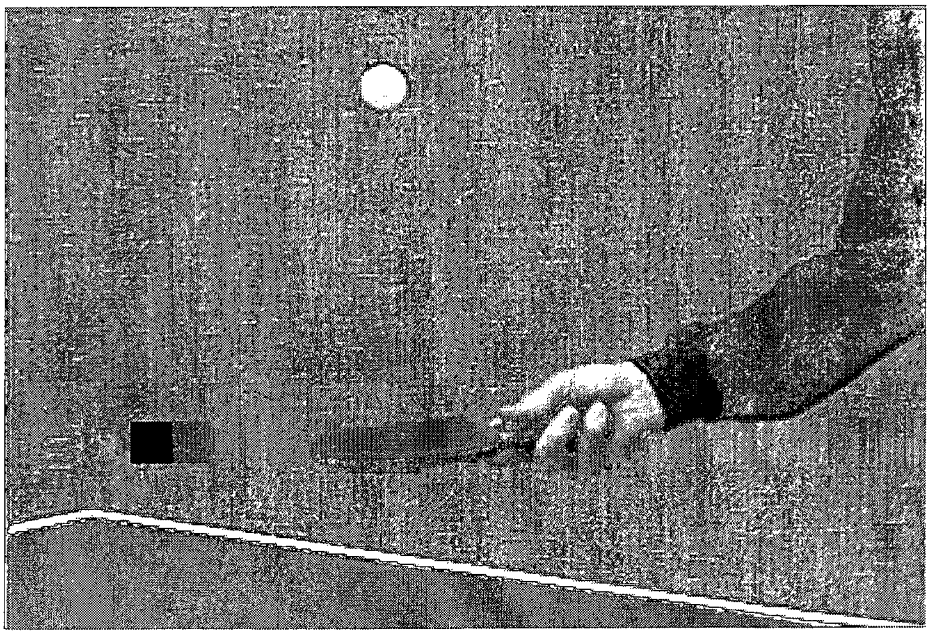


Figure 4.31: Frame 4 of video sequence "Table-Tennnis" decoded by the IJSCD-VVC scheme at channel SNR of 1.5dB and at the 4th iteration.

Figures below show decompressed results at frame 42.

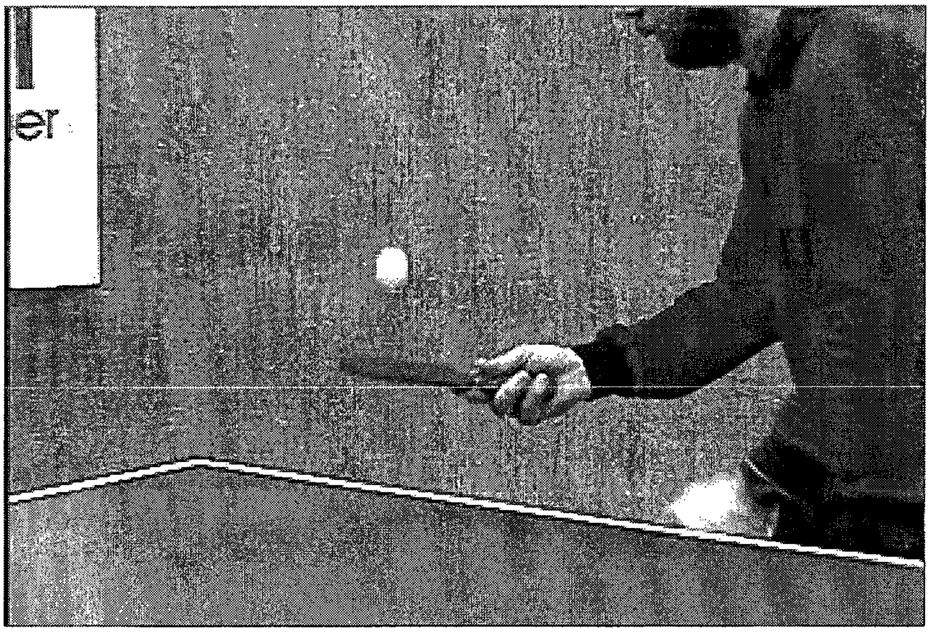


Figure 4.32: Decompressed error-free frame 42 of video sequence “Table-Tennis”.

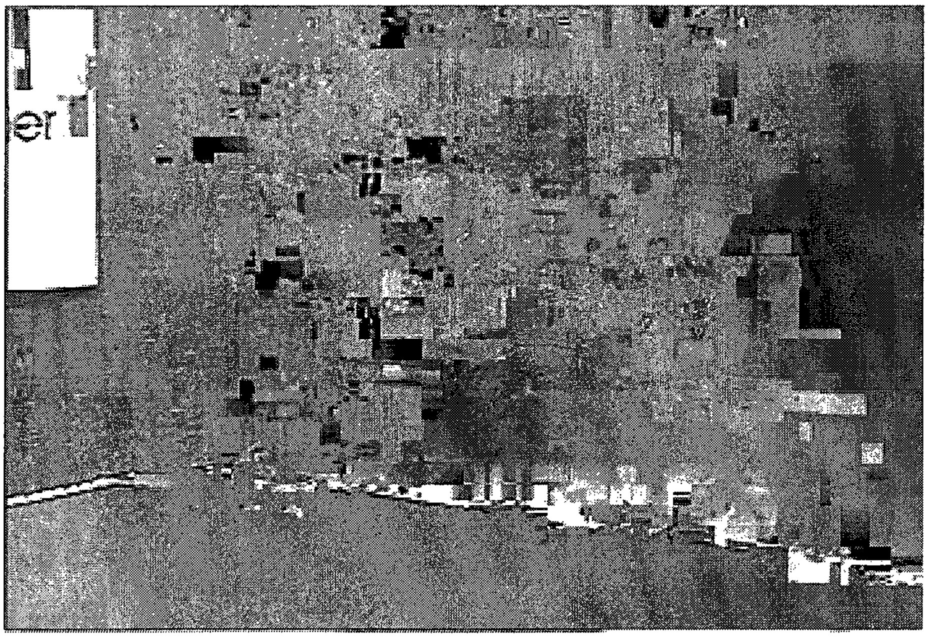


Figure 4.33: Frame 42 of video sequence “Table-Tennnis” decoded using convolutional decoding only at channel SNR of 1.5dB.

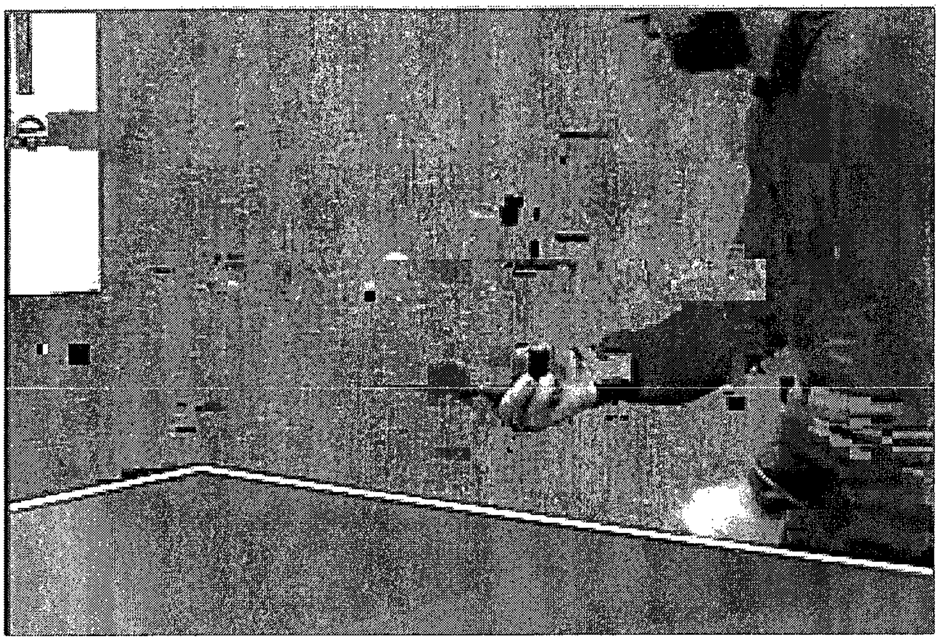


Figure 4.34: Frame 42 of video sequence “Table-Tennis” decoded by the IJSCD-I scheme at channel SNR of 1.5dB and at the 4th iteration.

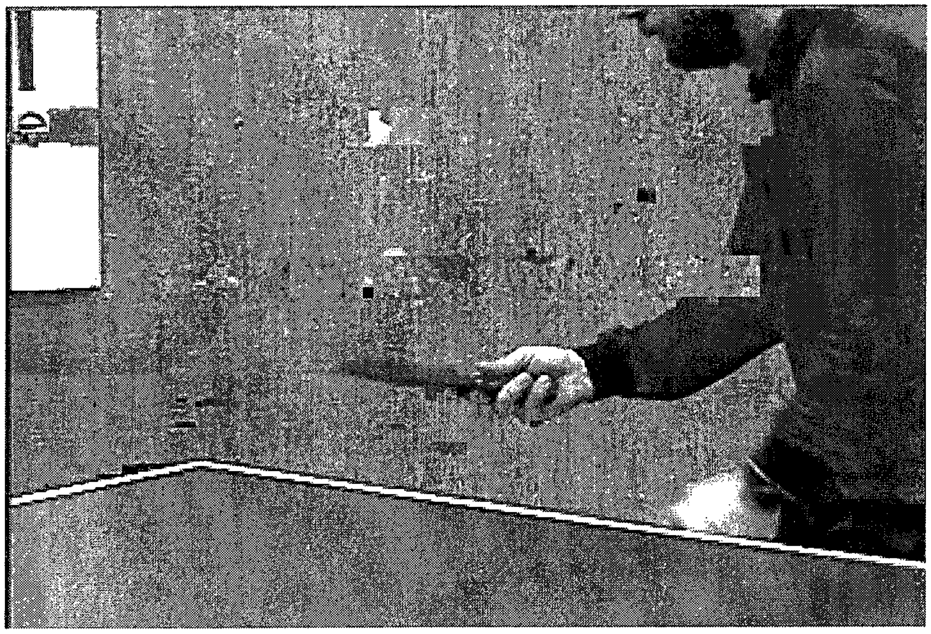


Figure 4.35: Frame 42 of video sequence “Table-Tennis” decoded by the IJSCD-VC-PI scheme at channel SNR of 1.5dB and at the 4th iteration.

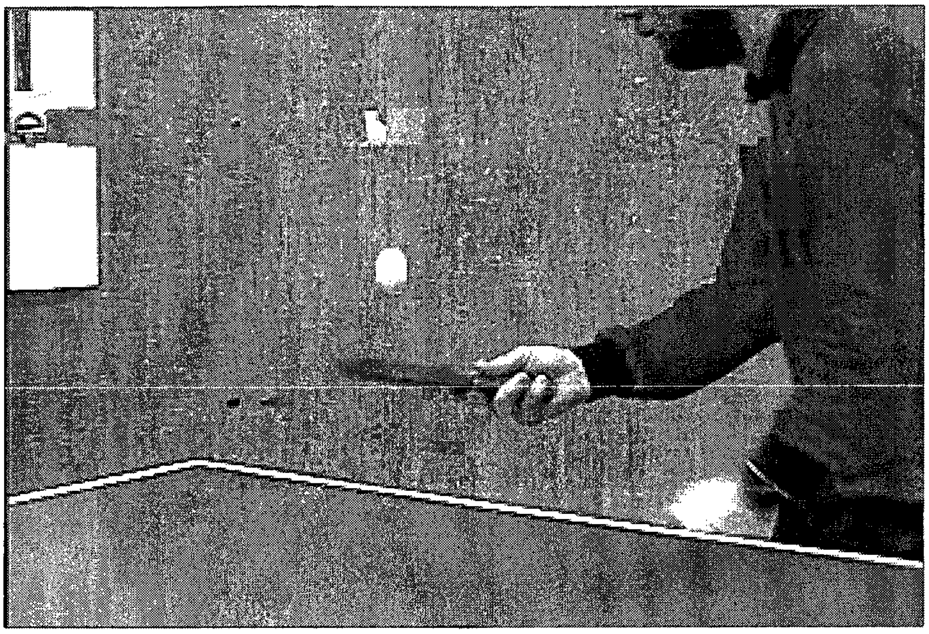


Figure 4.36: Frame 42 of video sequence "Table-Tennnis" decoded by the IJSCD-VVC scheme at channel SNR of 1.5dB and at the 4th iteration.

4.4.1.3 Simulation Results for Video "Garden"

In this section, the performance of the proposed scheme IJSCD-VVC and the IJSCD-I scheme is compared. The input video is "Garden".

4.4.1.3.1 Simulation Results for the IJSCD-I Scheme

Figure 4.37 and Figure 4.38 show the objective performance of the IJSCD-I scheme in terms of PSNR and BER for video "Garden" with 3 iterations. Results are also compared to the results of the convolutional coding scheme.

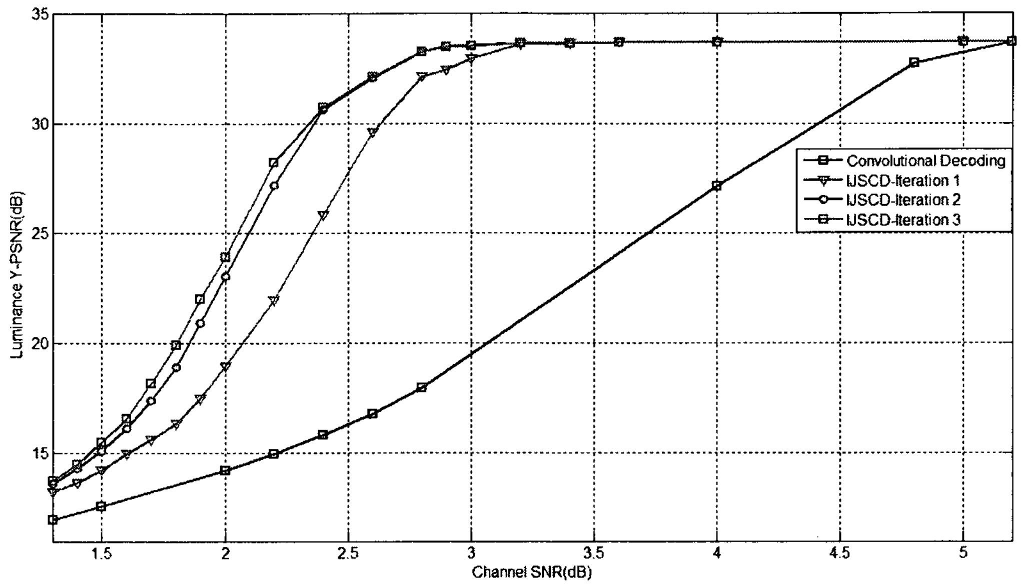


Figure 4.37: Y-PSNR vs. channel SNR for video “Garden” of the IJSCD-I scheme.

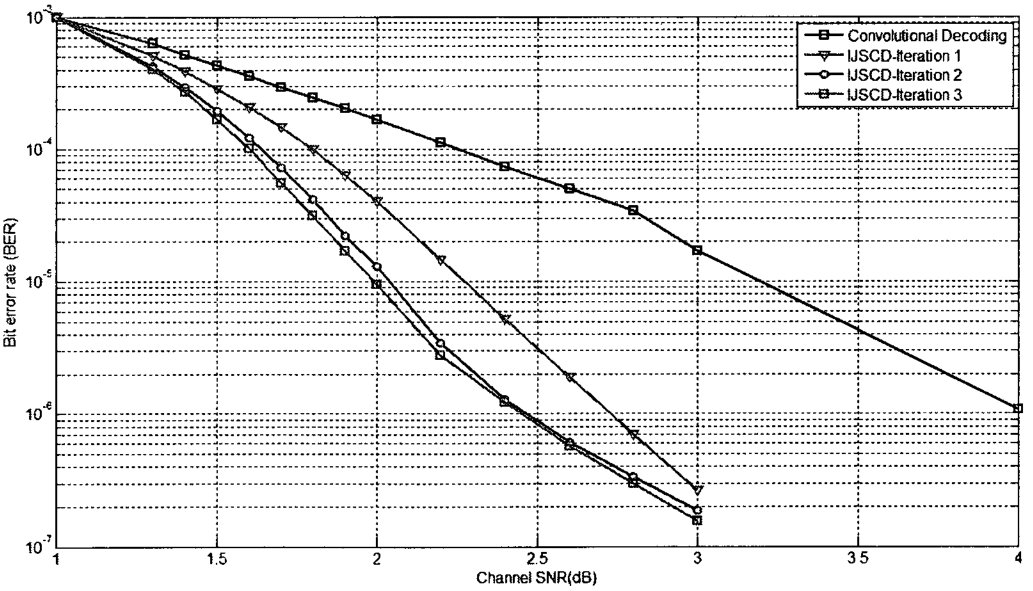


Figure 4.38: BER vs. channel SNR for video “Garden” of the IJSCD-I scheme.

4.4.1.3.2 Simulation Results for the Proposed Scheme IJSCD-VVC

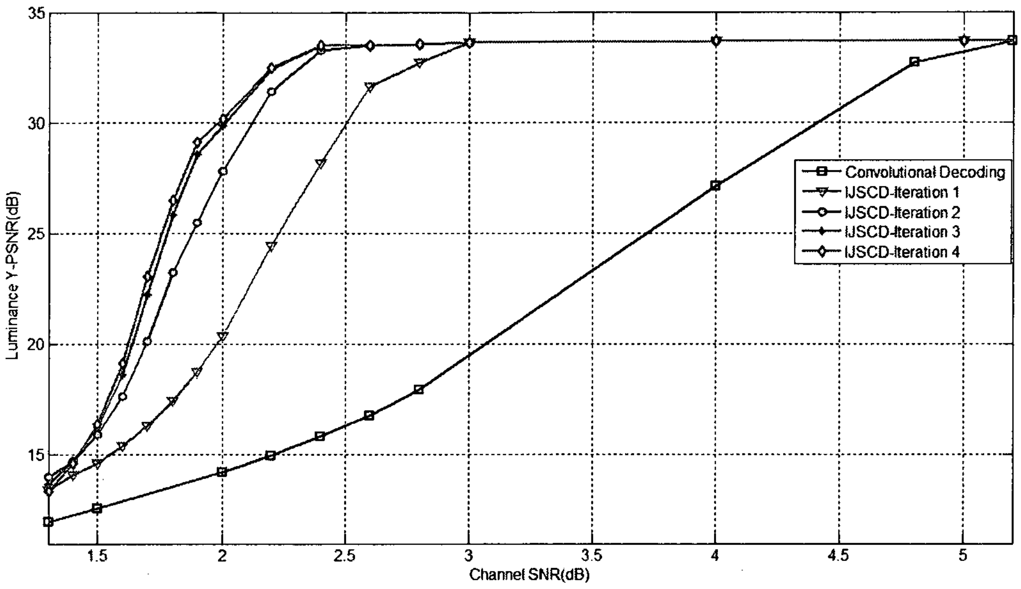


Figure 4.39: Y-PSNR vs. channel SNR for video "Garden" of the IJSCD-VVC scheme.

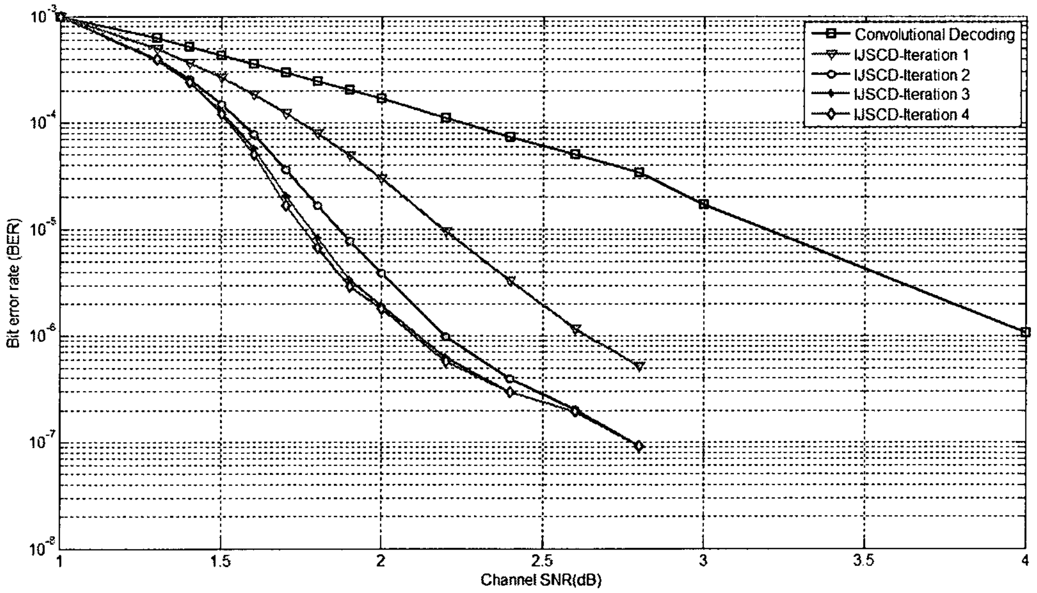


Figure 4.40: BER vs. channel SNR for video "Garden" of the IJSCD-VVC scheme.

Figure 4.39 and Figure 4.40 show the objective performance of the IJSCD-VVC scheme in terms of PSNR and BER for video "Garden" with 4 iterations.

4.4.1.3.3 Compare Performance of the Two Schemes at the 3th iteration

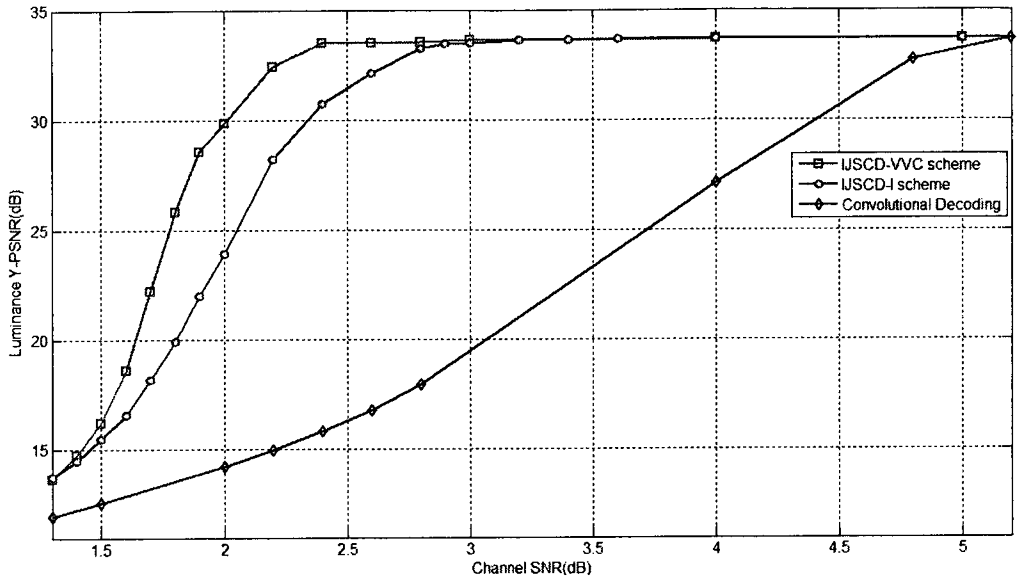


Figure 4.41: Comparison of the IJSCD-I scheme and the IJSCD-VVC scheme: Y-PSNR vs. channel SNR obtained at the 3th for video "Garden".

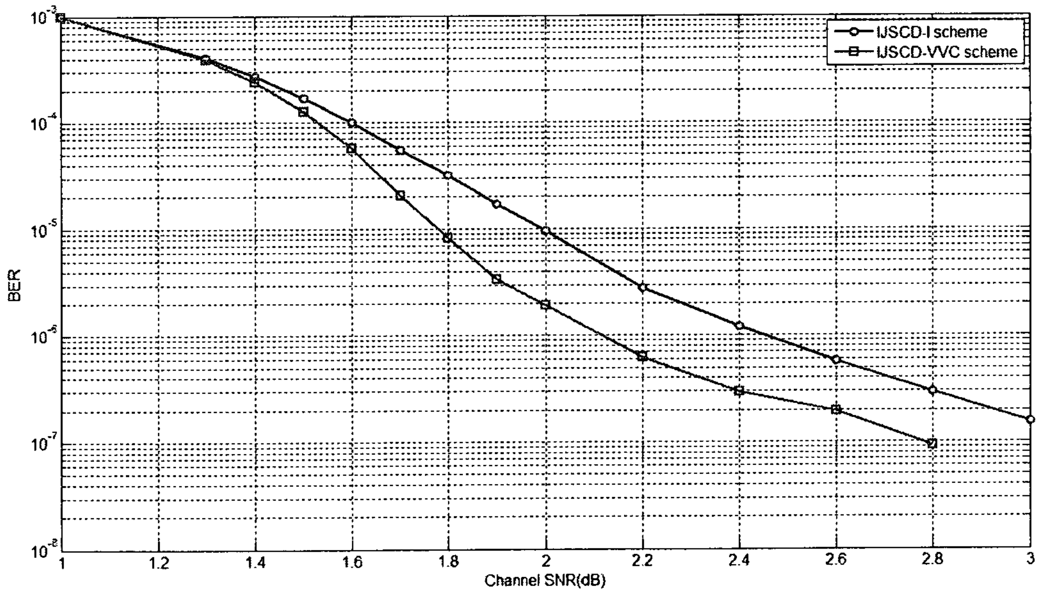


Figure 4.42: Comparison of the IJSCD-I scheme and the IJSCD-VVC scheme: BER vs. channel SNR obtained at the 3th for video "Garden".

Figure 4.41 and Figure 4.42 compare the performance of the two schemes in terms of PSNR and BER at the 3th iteration. One can see that, the maximum achievable PSNR is achieved at channel SNR of 2.8 dB when the IJSCD-I scheme is used. Meanwhile, the IJSCD-VVC scheme achieves that value at channel SNR of 2.4dB. With convolutional decoding only, the maximum achievable PSNR is obtained at the channel SNR of 5.2dB or higher. In other words, without bandwidth expansion, the IJSCD-VVC scheme can achieve 0.4dB of channel SNR reduction over the IJSCD-I scheme, and up to 2.8dB of channel SNR reduction over convolutional decoding.

4.4.2 Complexity Measurement

The complexity of the proposed scheme IJSCD-VVC is evaluated for one iteration using the time ratio as a measure as in [4]. The time ratio of a module in the scheme compares the run time of that module to the run time of the H.264 Decompressor. The time ratio for one IJSCD iteration is the sum of the time ratios of the MAP Decoder, the Deinterleaver, the Stream Merger, the SCG, the SSV, the Modifier and the Interleaver.

An experiment is performed to measure the time ratio for each of the modules in the proposed scheme. Video sequence "Football" is used in the experiment, which is compressed under the same conditions as in Section 4.4.1. The experiment is performed by running the IJSCD-VVC scheme three times at several channel SNR values and with the average being taken.

Simulation results show that, with the exception of the SSV, the complexity of all other modules in the IJSCD-VVC scheme is not dependent on the channel SNR. On the other hand, as discussed in Chapter 3, the complexity of the SSV is dependent on the channel SNR. This is because the lower the channel SNR, the more the number of errors that each slice has. Consequently, the number of slice candidates verified is in general larger at lower channel SNR. As a result, the complexity of the SSV is higher at lower channel SNR values.

Table 4.6: Time ratio between each module in the IJSCD-VVC scheme and the H.264 Decompressor for video "Football" at 2.0 dB channel SNR for the 1st iteration

Module	Time Ratio	% of Iteration Time
MAP Decoder	0.87	48.58%
Deinterleaver	0.0247	1.38%
Stream Merger	0.0004	0.022%
SCG	0.1147	6.4%
SSV	0.6723	37.5%
Modifier	0.084	4.7%
Interleaver	0.0247	1.38%
TOTAL	1.7908	100%

Table 4.6 shows the time ratio of each of the modules in the IJSCD-VVC scheme for video "Football" at 2.0dB channel SNR for the first iteration. The percentage of the iteration time that each module takes is also shown in Table 4.6. One can see that the majority of run time is spent between the MAP decoder and the SSV (about 86% of the run time). For the subsequent iterations, the complexity of the IJSCD-VVC scheme is reduced. This is because for subsequent iterations, BSCs are likely to be found sooner due to the reduction of the number of errors in the video. Consequently, the complexity of the SSV is reduced. One should also note that on the last iteration, the complexity of the H.264 Decompressor is taken into account (the time ratio of the H.264 Decompressor is always 1) while the time ratio of the Modifier and Interleaver are excluded (they are not run at the last iteration).

4.5 Summary

This chapter presents the IJSCD-VVC scheme for the transmission of H.264 compressed video using CABAC entropy coding. A feedback loop is created between the Source Decoder and the Channel Decoder, which allows iterative decoding to be performed.

The objective and subjective performance results show that, without bandwidth expansion, a single iteration of the proposed scheme IJSCD-VVC significantly outperforms convolutional decoding only. Furthermore, the performance of the IJSCD-VVC scheme is better than that of the IJSCD-I scheme at the same iteration. The complexity of the proposed scheme is also measured for a single iteration.

Chapter 5
Conclusion

5.1 Contributions

This thesis proposes two error resilient transmission schemes for H.264 compressed videos using CABAC entropy coding. Slice candidates that failed semantic verification are used in these schemes. And each scheme uses them as different ways.

First, Iterative Joint Source-Channel Decoding using Virtual Checking Method (IJSCD-VC) is proposed [21], which combines source decoding and channel decoding in an iterative manner. In this scheme, soft values of bits produced by the channel decoder are used to generate a list of slice candidates for each slice, which are ranked in descending order of likelihood. The SSV verifies semantics for these slice candidates to choose the BSC for each slice. A new semantic checking method, called Virtual Checking (VC), is proposed, which uses information of slice candidates that failed semantic verification to virtually check the current slice candidate. Simulation results show that, using VC can reduce the computational complexity while keeping the same performance (IJSCD-VC-FS Scheme). Alternatively, using VC yields a performance improvement over the scheme without using VC while keeping almost the same computational complexity (IJSCD-VC-PI Scheme).

Second, Iterative Joint Source-Channel Decoding using Voting and Virtual Checking Method (IJSCD-VVC) is proposed, in which a new Modifier is proposed to modify soft values of bits at the source decoder. In particular, when the BSC passes verification, the proposed Modifier enlarges soft values of bits corresponding to their hard values in the BSC. When all slice candidates fail verification, the proposed Modifier applies the Voting method, in which several slice candidates that failed semantic verification follow the set up Voting rule to give each flip-bit a vote. The voting results of flip-bits are used to modify their soft values. After the modification, soft values are fed back to the channel decoder for the next iteration. Simulation results showed that, for video sequence "Table-Tennis" for instance, without bandwidth expansion, the IJSCD-VVC scheme can achieve 0.5dB of channel SNR reduction over the IJSCD-I scheme proposed in [5], and up to 2.8dB of channel SNR reduction over convolutional decoding.

5.2 Conclusions

According to the work in this thesis, the following conclusions can be drawn.

- The semantic correctness is a good measure to evaluate the correctness of a slice for H.264 compressed video using CABAC entropy coding. Thus, it can be applied at the source decoder to correct transmission errors.
- Many slice candidates (which are generated for a slice) fail semantic verification in the same way (i.e. they have the same DB). Thus, the semantic verification results of previously verified slice candidates can be used to check the semantic correctness of the currently considered slice candidate. This speeds up the semantic verification process.
- The information of the slice candidates that failed semantic verification can be used to evaluate the correctness of several bits.

5.3 Future Works

The channel code used in this thesis is a convolutional code. In the future, the IJSCD-VVC scheme could be implemented using a turbo code, which is a more powerful channel code that is able to further enhance error resilience of compressed video when transmitted over noisy channels.

In the Voting Method, to modify the soft value of a bit, each slice candidate uses the Voting Rule to give it a vote. The Voting Rule proposed in this thesis does not seriously take into account the interaction between flip-bits in the slice. As mentioned in Chapter 4, the interaction between flip-bits in a slice sometimes makes a vote unreliable. In the future, the Voting Rule could be improved that handle the interaction of flip-bits in a slice.

There are some other jobs that can be parts of the future work such as considering a fading channel instead of an AWGN channel and using different compressed scheme such as H.264/MPEG-4 SVC (Scalable Video Coding).

REFERENCES

[1] I. E. G. Richa rson, "H.264/MPEG-4 Part 10," in *H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia*, West Sussex, England: Wiley, 2003, pp. 159-222.

[2] T. Wiegand, H. Schwa rz, A. Joch, F. Kossentini , and G. J. Sullivan, "Rate-constrained coder control and comparison of video coding standards," *IEEE Trans. Circuits Syst. Video Techno.*, vol. 13, no. 7, pp. 688-703, July 2003.

[3] Y. Wang, S. Wenger, J. Wen, and A. K. Katsaggelos, "Error resilient video coding techniques," *IEEE Signal Processing Mag.*, vol. 17, pp. 61–82, July 2000.

[4] D. Levine, "Error resilient transmission of H.264 compressed video using CABAC entropy coding," M.A.Sc. thesis, Dept. Elect. Comput. Eng., Concordia Univ., Montreal, Quebec, 2007.

[5] D. Levine, W. E. Lynch, and T. Le-Ngoc, "Iterative joint source-channel decoding of H.264 compressed video," in *IEEE Int. Symp. Circuits and Systems*, pp. 1517-1520, May 2007.

[6] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits Syst. Video Techno.*, vol. 13, no. 7, pp. 560-576, July 2003.

[7] D. Marpe, H. Schwarz, and T. Wiegand, "Context-based adaptive binary arithmetic coding in the H.264/AVC video compression standard," *IEEE Trans. Circuits Syst. Video Techno.*, vol. 13, no. 7, pp. 620-636, July 2003.

[8] Joint Video Team (JVT), "Recommendation ITU-T H.264: Advanced video coding for generic audiovisual services," ITU-T, May 2003.

[9] H.264/AVC Codec Software Archieve. Available online: http://iphome.hhi.de/suehring/tml/download/old_jm/.

[10] X. Tingjun, W. Xuan, and L. Jianhua, "A joint source-channel coding scheme using low-density parity-check codes and error resilient arithmetic codes," in *Wireless Communications and Networking Conf.*, pp. 672-676, March 2007.

[11] B. D. Pettijohn, M. W. Hoffman, and K.Sayood, "Joint source/channel coding using arithmetic codes," *IEEE Trans. Commun.*, vol. 49, Issue 5, pp. 826 - 836, May 2001.

[12] M. Grangetto, P. Cosman, and G. Olmo, "Joint source/channel coding and MAP decoding of arithmetic codes," *IEEE Trans. Commun.*, vol. 53, Issue 6, pp. 1007 - 1016, June 2005.

[13] Y. Wang and S. Y. Yu, "Joint source-channel decoding for H.264 coded video stream," *IEEE Trans. Consum. Electron.*, vol. 51, Issue 4, pp. 1273-1276, Nov. 2005.

[14] V. Papadakis, W. E. Lynch, and T. Le-Ngoc, "Syntax based error concealment," in *Proc. IEEE Int. Symp. Circuits and Systems*, vol. 4, pp. 265-268, 1998.

[15] W. E. Lynch, V. Papadakis, R. Krishnamurthy, and T. Le-Ngoc, "Syntax and discontinuity based error concealment," in *Proc. IEEE Int. Symp. Circuits and Systems,* vol. 4, pp. 235-238, May 1999.

[16] Y. Mei, W. E. Lynch, and T. Le-Ngoc, "Joint forward error correction and error concealment for compressed video," in *Int. Conf. Information Technology: Coding and Computing*, pp. 410-415, April 2002.

[17] W. E. Lynch and T. Le-Ngoc, "Syntax and discontinuity based error concealment for compressed video in a packet environment," in *Int. Conf. Information Technology: Coding and Computing*, pp. 258-262, April 2001.

[18] Z. Peng, Y. F. Huang, and D. J. Costello, "Turbo codes for image transmission - a joint channel and source decoding approach," *IEEE J. Sel. Areas Commun.*, vol. 18, no. 6, pp. 868-879, June 2000.

[19] L. Pu, Z. Wu, A. Bilgin, M. W. Marcellin, and B. Vasic, "Iterative joint source/channel decoding for JPEG2000," in *Conf. Rec. Thirty-Seventh Asilomar Conf. Signals, Systems and Computers*, vol. 2, pp. 1961-1965, Nov. 2003.

[20] X. F. Ma and W. E. Lynch, "Iterative joint source-channel decoding using turbo codes for MPEG-4 video transmission," in *Int. Conf. Acoustics, Speech and Signal Processing*, vol. 4, pp. 657-660, 2004.

[21] N. Q. Nguyen, W. E. Lynch, and T. Le-Ngoc, "Iterative joint source-channel decoding for H.264 video transmission using virtual checking method at the source decoder," in *Canadian Conf. Electrical and Computer Engineering 2010*, to be published.

[22] Y. Wang and Q. F. Zhu, "Error control and concealment for video communication: a review," *Proc. IEEE,* vol. 86, Issue 5, pp. 974-997, 1998.

[23] T. P. Chen and T. Chen, "Second-generation error concealment for video transport over error prone channels," in *Proc. International Conf. Image Processing*, vol. 1, pp. I.25-I.28, Sept. 2002.

[24] W. Zeng and B. Liu, "Geometric structured based error concealment with novel applications in block-based low-bit-rate coding," *IEEE Trans. Circuits Syst. Video Techno.*, vol. 9, Issue 4, pp. 648-665, 1999.

[25] M. Friebe and A. Kaup, "Fading techniques for error concealment in block-based video decoding systems," *IEEE Trans. Broadcast.*, vol. 53, no. 1, pp. 286-296, Mar. 2007.

[26] W. M. Lam, A. R. Reibman and B. Liu, "Recovery of lost or erroneously received motion vectors," in *IEEE Int. Conf. Acoustics, Speech, and Signal Processing*, vol. 5, pp. 417-420, Mar.1993.

[27] S. Lee, D. Choi, and C. Hwang, "Error concealment using affine transform for H.263 coded video transmissions," *IEEE Electron. Lett.*, vol. 37, Issue 3, pp. 218-220, 2001.

[28] B. Yan and K. W. Ng, "A novel selective motion vector matching algorithm for error concealment in MPEG-4 video transmission over error-prone channels," *IEEE Trans. Consum. Electron.*, vol. 49, Issue 4, pp. 1416-1423, Nov. 2003.

[29] Y. Chen, Y. Hu, O. C. Au, H. Li, and C. W. Chen, "Video error concealment using spatial-temporal boundary matching and partial differential equation," *IEEE Trans. Multimedia*, vol. 10, pp. 2-15, Jan. 2008.

[30] G. L. Wu and S. Y. Chien, "Spatial-temporal error detection scheme for video transmission over noisy channels," in *Ninth IEEE Int. Symp. Multimedia*, pp. 78-85, Dec. 2007.

[31] S. B. Lee, T. J. Kim, J. W. Suh, and H. D. Bae, "Error concealment for 3G-324M mobile videophones over a WCDMA networks," in *Int. Conf. Consumer Electronics,* pp. 1-2, Jan. 2007.

[32] J. Y. Pyun and H. J. Choi, "Error concealment aware error resilient video coding over wireless burst-packet-loss network," in *5th IEEE Consumer Communications and Networking Conf.*, pp. 824-828, 2008.

[33] W. Zhu, Y. Wang, and Q. F. Zhu, "Second-order derivative-based smoothness measure for error concealment," *IEEE Trans. Circuits Syst. Video Techno.*, vol. 8, Issue 6, pp. 713-718, 1998.

[34] S. Valente, C. Dufour, F. Groliere, and D. Snook, "An efficient error concealment implementation for MPEG-4 video streams," *IEEE Trans. Consum. Electron.*, vol. 47, pp. 568-578, Aug. 2001.

[35] Z. Ji, L. Feng, S. Hui, and W. Gang, "An effective error concealment framework for H.264 decoder based on video scene change detection," in *Fourth Int. Conf. Image and Graphics*, pp. 285-290, 2007.

[36] T. H. Wu, G. L. Wu, C. Y. Chen, and S. Y. Chien, "Enhanced temporal error concealment algorithm with edge-sensitive processing order," in *IEEE Int. Symp. Circuits and Systems,* pp. 3466-3469, May 2008.

[37] W. Y. Kung, C. S. Kim, and C. C. J. Kuo, "Spatial and temporal error concealment techniques for video transmission over noisy channels," *IEEE Trans. Circuits Syst. Video Techno.*, vol. 16, no. 7, pp. 789-802, July 2006.

[38] R. H. M. Zaragoza, "Binary convolutional codes," in *The Art of Error Correcting Coding,* West Sussex, England: Wiley, 2002, pp. 75-100.

[39] L. Hanzo, T. H. Liew, and B. L. Yeap, "Turbo convolutional coding," in *Turbo Coding, Turbo Equalisation and Space-Time Coding for Transmission over Fading Channels,* West Sussex, England: Wiley, 2002, pp. 107-171.

[40] X. F. Ma, "Iterative joint source and channel decoding using turbo codes for MPEG-4 video transmission," M.A.Sc. thesis, Dept. Elect. Comput. Eng., Concordia Univ., Montreal, Quebec, 2004.

[41] D. Levine, W. E. Lynch, and T. Le-Ngoc, "Observations on error detection in H.264," in *50th Midwest Symp. Circuits and Systems*, August 2007.

[42] R. E. Walpole and R. H. Myers, "One- and two-sample estimation problems," in *Probability and Statistics for Engineers and Scientists,* 5th ed. New York: Macmillan, 1993, pp. 239-286.

[43] R. Talluri, "Error-resilient video coding in the ISO MPEG-4 standard," *IEEE Commun. Mag.*, vol. 6, Issue 36, pp. 112-119, June 1998.

[44] M. Wien, "Variable block-size transform for H.264/AVC," *IEEE Trans. Circuits Syst. Video Techno.*, vol. 13, no. 7, pp. 604-613, July 2003.

[45] H. S. Malvar, A. Hallapuro, M. Karczewicz, and L. Kerofsky, "Low-complexity transform and quantization in H.264/AVC," *IEEE Trans. Circuits Syst. Video Techno.*, vol. 13, no. 7, pp. 598-603, July 2003.

[46] M. Wien, "Variable block-size transform for H.264/AVC," *IEEE Trans. Circuits Syst. Video Techno.*, vol. 13, no. 7, pp. 604-613, July 2003.

[47] J. Hagenauer, E. Offer, and L. Papke, "Iterative coding of binary block and convolutional codes," *IEEE Trans. Inf. Theory,* vol. 42, Issue 2, pp. 429-445, Mar. 1996.

[48] B. Katz, S. Greenberg, N. Yarkoni, N. Blaunstien, and R. Giladi, "New error-resilient scheme based on FMO and dynamic redundant slices allocation for wireless video transmission," *IEEE Trans. Broadcast.*, vol. 53, Issue 1, pp. 308-319, Mar. 2007.

[49] Y. Guo, Y. K. Wang, and H. Li, "Error resilient mode decision in scalable video coding," in *IEEE Int. Conf. Image Processing*, pp.2225-2228, 2006.

[50] B. Yan and K. W. Ng, "An improved unequal error protection technique for the wireless transmission of MPEG-4 video," in *Int. Conf. Information,*

Communications and Signal Processing and the 4th Pacific Rim Conf. Multimedia, pp. 513-517, 2003.

[51] P. Y. Yip, J. A. Malcolm, W. A. C. Fernando, K. K. Loo, and H. K. Arachchi, "Joint source and channel coding for H.264 compliant stereoscopic video transmission," in *Canadian Conf. Electrical and Computer Engineering*, pp. 188-191, May 2005.

[52] K. Ali and F. Labeau, "Joint source-channel turbo decoding of entropy coded sources," *Vehicular Technology Conf.*, vol. 3, pp. 1960-1964, Sep. 2005.

[53] Y. Liu, Q. S. Tong, A. D. Men, Z. Y. Quan, and B. Yang, "A joint source-channel coding scheme focused on unequal error protection for H.264 transmission over MIMO-OFDM system," in *Int. Colloq. Computing, Communication, Control, and Management*, pp. 491-495, Aug. 2008.

[54] M. Bystrom and J. W. Modestino, "Combined source-channel coding schemes for video transmission over an additive white Gaussian noisy channel," *IEEE J. Sel. Areas Commun.*, vol. 18, Issue 6, pp. 880-890, June 2000.

[55] 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, pp. 971-980, Oct. 2003.

[56] J. Kliewer and R. Thobaben, "Iterative joint source channel decoding of variable-length codes using residual source redundancy," *IEEE Trans. Wireless Commun.*, vol. 4, no. 3, pp. 919-929, May 2005.

[57] Q. Chen and K. P. Subbalakshmi, "Joint source-channel decoding for MPEG-4 video transmission over wireless channels," *IEEE J. Sel. Areas Commun.*, vol. 21, Issue 10, pp. 1780-1789, Dec. 2003.

[58] K. Lakovic, T. Tian, and J. Villasenor, "Iterative decoder design for joint source-channel LDPC decoding," in *Int. Conf. Computer as a Tool*, vol. 1, pp. 486-489, Nov. 2005.

[59] A. Elbaz, R. Pyndiah, B. Solaiman, and O. A. Sab, "Iterative decoding of product codes with a priori information over a Gaussian channel for still image transmission," in *Global Telecommunications Conf.*, vol. 5, pp. 2602-2606, 1999.

[60] Nasruminallah and L.Hanzo, "Convergence behaviour of iteratively decoded short block-codes in H.264 joint source and channel decoding," in *Vehicular Technology Conf.*, pp. 1-5, April 2009.

[61] L. Xu, M. W. Hoffman, and K. Sayood, "Hard decision and iterative joint source channel coding using arithmetic codes," in *Data Compression Conf.*, pp. 203-212, Mar. 2005.