Iterative Joint Source and Channel Decoding Using Turbo Codes for MPEG-4 Video Transmission

Xiao Feng Ma

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 at

Concordia University

Montréal, Québec, Canada

February 2004

Your file Votre référence
ISBN: 0-612-91076-8
Our file Notre référence
ISBN: 0-612-91076-8

ABSTRACT

Iterative Joint Source and Channel Decoding Using Turbo Codes for MPEG-4 Video Transmission

Xiao Feng Ma

This thesis presents a novel iterative joint source and channel decoding scheme using turbo codes for MPEG-4 video transmission over noisy channels. The proposed scheme, on one hand, utilizes the channel soft outputs generated by a turbo decoder to assist syntax based error concealment in a source decoder. On the other hand, the residual redundancy extracted by the source decoder is fed back to the channel decoder through modifying the extrinsic information exchanged between the two constituent MAP decoders of the turbo decoder so as to improve the error performance of the turbo decoder. With video packet mixer, the proposed scheme can correct most of the turbo coding blocks with a large number of bit errors. Simulation results show significant improvement in terms of BER, PSNR and the reconstructed video quality.

Keywords: Joint source and channel decoding, turbo codes, MPEG-4, video transmission, error concealment, error resilient, syntax.

To my mother, my husband and the memory of my father

ACKNOWLEDGEMENTS

First, I would like to thank my supervisor Dr. William E. Lynch for proposing the research topic and directing this work. I would also like to thank all the professors with whom I have interacted during my study at Concordia University. Last but not least, I would like to thank my colleague Mr. Cheng-Yu Pai and other friends at Concordia for their support and for sharing with me their experience.

This work has been supported by the Natural Science and Engineering Research Council of Canada.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$L(u_k)$	LAPP ratio or soft output from a MAP decoder
N	Interleaver length
N_f	Number of flip bit candidates
N_{VP}	Number of VP candidates
M_A	Overall discontinuity measure
N_m	Number of bits from the same video packet in a turbo coding block
$\frac{E_b}{N_0}$	Channel SNR
r	Code rate
E_b	Energy of a bit
N_0	One sided noise power spectral density
k	Time index
u_k	Transmitted bit at time k
$s,\ s'$	State of the RSC encoder
$\mathbf{y}$	Received noisy bit sequence
$\alpha_k(s)$	Forward state metric
$\beta_k(s')$	Backward state metric
$\gamma_k(s', s)$	Branch metric
N_0	One sided noise power spectral density
$P_{diff}(u_k)$	Logarithm probability difference of bit u_k
$P(u_k\|\mathbf{y})$	APP of bit u_k
$P(\text{VPC}_i\|\mathbf{y})$	APP of VPC i

LIST OF ACRONYMS

APP	*a posteriori* Probability
APRI-SOVA	*a priori* Soft Output Viterbi Algorithm
AWGN	Additive White Gaussian Noise
BER	Bit Error Rate
BPSK	Binary Phase Shift Keying
DCT	Discrete Cosine Transform
DVB	Digital Video Broadcasting
DVD	Digital Versatile Disc
EC	Error Concealment
ED	Error Detection
ER	Error Resilience
FBC	Flip Bit Candidate
FEC	Forward Error Correction
FLC	Fixed Length Coding
GOV	Group of VOP
HDTV	High-Definition TV
HEC	Header Extension Code
IDCT	Inverse Discrete Cosine Transform
IEC	International Electrotechnical Commission
IJSCD	Iterative Joint Source and Channel Decoding
ISO	International Organization for Standardization
LAPP	Logarithm *a posteriori* Probability
MAD	Mean Absolute Difference
MAP	Maximum *a posteriori*
MB	Macroblock
MM	Modification Module

MPEG	Moving Picture Experts Group
MSE	Mean Square of Error
MVPCG	Multiple Video Packet Candidate Generator
NEB	Number of Error Bits
NSC	Nonsystematic Convolutional Code
PSNR	Peak Signal-to-Noise Ratio
RSC	Recursive Systematic Convolutional
RVLC	Reversible VLC
SBEC	Syntax Based Error Concealment
SBECTC	Syntax Based Error Concealment using Turbo Codes
SCA	Smallest Combination Algorithm
SDBEC	Syntax and Discontinuity Based Error Concealment
SNR	Signal to Noise Ratio
TD	Turbo Decoding
VHS	Video Home System
VLC	Variable Length Coding
VO	Video Object
VOL	Video Object Layer
VOP	Video Object Plane
VP	Video Packet
VPC	Video Packet Candidate
VPDM	VP De-Mixer
VPM	VP Mixer
VS	Video Sequence

Chapter 1

Introduction

1.1 Transmission of Compressed Videos

Video transmission has been a major issue in today's communication networks. In video transmission, many challenges have been encountered mainly due to limited bandwidth and noisy channels. Video compression techniques are developed to deal with the problem of limited bandwidth. However, these compression techniques make the noisy channel problem worse and thus robust video transmission schemes are necessary to combat channel noise and to provide videos with acceptable quality [1].

Video compression is essential in a video transmission system because the available bandwidth is restricted. Bandwidth is in general measured in bits per second. In the real world, there's always a restriction on the available bandwidth. For example, the available bandwidth in regular telephone lines is 33.6 kbits/s and 20 Mbits/s in digital television system in North America [2]. A 1-second video with frame rate 30 frames/s and picture size 352×240 in RGB color space needs $352 \times 240 \times 3 \times 30 \times 8 = 60,825,600$ bits to represent it if each pixel is represented using 8 bits. Hence, without compression, it's impossible to transmit videos over most of the current communication networks. Not only does video compression make transmission possible, but it also enables efficient use of the available bandwidth. Since bandwidth costs money, video compression reduces the cost of transmission for both the video providers and consumers.

Common video compression techniques exploit the spatial and temporal redundancy of videos. The compression standards produced by Moving Pictures Experts Group (MPEG) including MPEG-1 (ISO/IEC 11172), MPEG-2 (ISO/IEC 13818) and MPEG-4 (ISO/IEC 14496) have been accepted worldwide and have given rise to a wide range of applications. Notably, MPEG-2 has been chosen as the compression scheme for both the European Digital Video Broadcasting (DVB) and Digital Versatile Discs (DVDs) [3]. MPEG-4 known as the standard for multimedia applications is the newest standard on compression. The work in this thesis follows

the MPEG-4 standard. The compression techniques in the MPEG-4 standard include Discrete Cosine Transform (DCT), Variable Length Coding (VLC), motion estimation/compensation, etc. The DCT is aimed to reduce spatial redundancy in neighboring pixels of a frame and motion estimation/compensation exploits temporal redundancy in adjacent frames. VLC is an efficient source coding method, which represents input symbols using different length codewords according to the symbols' probability.

When a compressed video is transmitted over communication channels, it suffers from channel noise. Channel induced errors can be divided into three categories according to channel condition [4, 1]: (1) Packet losses due to network congestion; (2) Burst errors resulted from multipath propagation in mobile wireless networks; (3) Random bit errors caused by imperfections of physical channels. In this thesis, an Additive White Gaussian Noise (AWGN) channel is assumed and thus only random bit errors are considered.

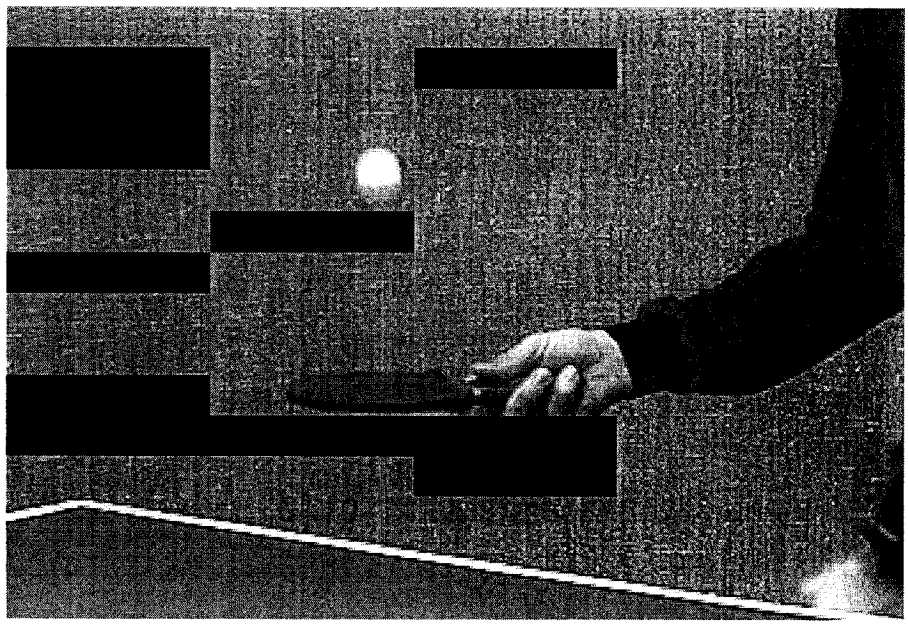

Figure 1.1: Error propagation due to bit errors in VLC codes. Frame 1 of "Table-Tennis" and the reference frame for Frame 2.

After compression, most of the redundancy in a video is removed. As a result,

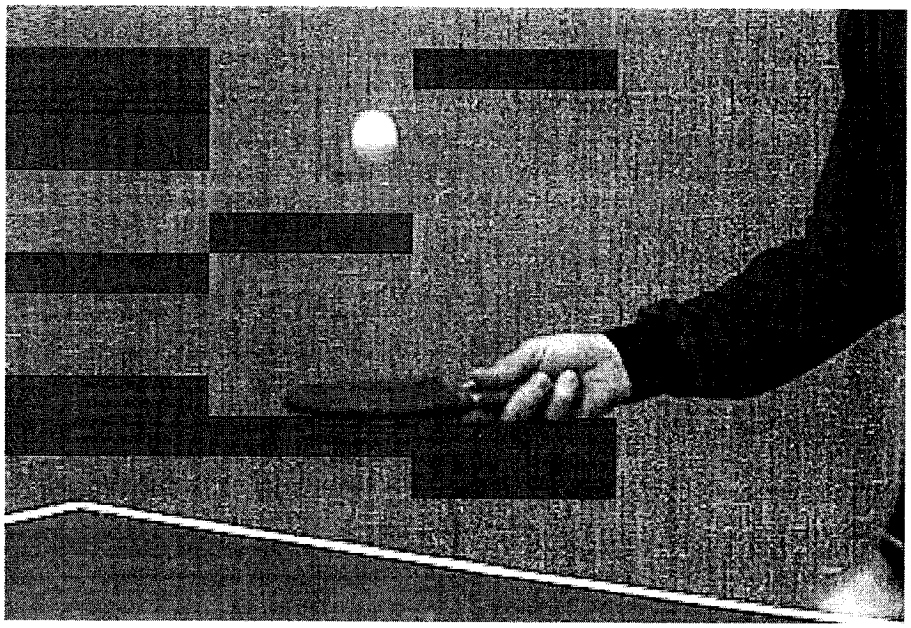

Figure 1.2: Error propagation from the reference frame, Frame 1, to motion compensated frame, Frame 2.

compressed video is more fragile to channel errors than uncompressed video in the sense that a few channel errors may cause severe quality degradation because of error propagation. For example, in VLC coding in the MPEG-4 standard, since there's no fixed codeword length, a single bit error could cause the length of a codeword to be wrongly assessed and then all subsequent codewords would also be wrongly decoded until synchronization could be re-established. The black strips in Figure 1.1 show the effect of bit errors that cause error propagation. In addition to error propagation caused by VLC coding, errors can also propagate in the temporal direction, due to motion estimation/compensation. Figure 1.2 shows the error propagation in temporal direction.

In order to combat channel noise, three kinds of techniques have been developed for robust video transmission [4, 1]: (1) Forward Error Correction (FEC) codes, which provide error protection for compressed videos from the viewpoint of channel coding; (2) Error Resilient (ER) schemes, which make compressed bitstream resilient to channel errors from the viewpoint of compression; (3) Error Concealment

(EC) schemes, which conceals the errors in a frame at the source decoder side. Some references on these three kinds of techniques can be found in [1]. In fact, these techniques are not mutually exclusive, but can be used jointly. This thesis studies a joint source and channel decoding scheme for error concealment in video decompression process, in which turbo codes are employed for error protection.

1.2 Problem Statement

In this thesis, a robust video transmission scheme, Iterative Joint Source and Channel Decoding (IJSCD) is investigated and developed. In this scheme, a powerful channel code, turbo code, is employed to protect the compressed bitstream, i.e., the output from an MPEG-4 compressor. Binary Phase Shift Keying (BPSK) is used for modulation and Additive White Gaussian Noise (AWGN) channel is assumed.

Since the IJSCD is a joint scheme of channel and source decoding, the interactivity between the source decoder and the channel decoder has to be established. Thus, the main problem of the IJSCD is two-fold. First, how can the source decoder make use of the information from the channel decoder, i.e., turbo decoder? Second, how can the information from the source decoder be fed back to the channel decoder? In addition, since the IJSCD is an iterative scheme, the third problem is that how the interactivity between the source decoder and the channel decoder can go on iteratively?

The information from the turbo decoder considered in this thesis includes the hard decision and soft output of each decoded bit. To solve the first problem stated above, the relationship between the hard decision and soft output is investigated. Based on this relationship, multiple Video Packet (VP) candidates are constructed for each VP (note that VP is the lowest syntax layer with a resynchronization code in a compressed MPEG-4 video). The *a posteriori* Probability (APP) of each VP candidate is also calculated based on the soft outputs from the turbo decoder. All

the VP candidates together with their APPs are sent to the source decoder. Note, for a BPSK system, the hard decision/output is 1 or 0 for each information/input bit, which is made by a channel decoder. If a decoder outputs a continuous number or chooses one from more than two numbers for each information/input bit, this decoder is said to make soft decisions/outputs.

The source decoder chooses the best VP candidate according to the APP of each candidate and the redundant information extracted from the compressed video. The information of the best VP candidate is fed back to the channel decoder by a modification scheme. In addition to feed back the information from the source decoder to the turbo decoder, the modification scheme is also developed in the way so that the interactivity between the source decoder and the channel decoder can work iteratively.

1.3 Figures of Merit

Two kinds of measurement have to be introduced since this thesis deals with robust video transmission over noisy channels. The first kind of measurement is related to channel coding/decoding. The other is how to evaluate the quality of reconstructed videos.

From the viewpoint of channel coding, the figures of merit are required such as channel SNR, Bit Error Rate (BER), Number of Error Bits (NEB) and code rate.

- Channel SNR is defined as

$$\text{Channel SNR} \triangleq \frac{E_b}{N_0} \tag{1.1}$$

where E_b represents the energy of a bit and N_0 the one sided noise power spectral density. Channel SNR is useful for evaluating the energy requirement of a digital transmission system [5].

- Number of Error Bits (NEB) is the absolute number of error bits in a received/reconstructed bitstream.

- Bit Error Rate (BER) is calculated by dividing NEB by the total number of bits in the compressed bitstream. BER and NEB can be used to evaluate the goodness of a received/reconstructed bitstream.

- Code rate r is defined as

$$r \triangleq \frac{n_I}{n_C} \tag{1.2}$$

  where n_I is the number of information bits and n_C represents the number of channel coded bits corresponding to n_I information bits. In a binary system, the channel-coded system requires a bandwidth expansion by a factor of $1/r$ to maintain the same bitrate as the uncoded system [6].

The quality of a reconstructed video also need to be assessed, which is generally done subjectively and objectively [7]. Subjective methods require a group of human viewers to rate the quality in a well-controlled environment, which can be a costly and time-consuming process. However, it yields accurate results since the end users of video are human beings. Objective methods don't use human viewers but rather evaluate the video automatically. These methods usually implement an algorithm which measure video quality based on the comparison of a source/original and a processed video [7].

In this thesis, rigorous subjective methods are not employed due to the consideration of cost and time. Instead, two alternative ways are used: 1) Some frames of an original and reconstructed video are presented for readers' own evaluation. 2) The author views the videos and gives the comments on their subjective quality.

Common objective methods of assessing video quality includes calculating Mean Square of Error (MSE), Mean Absolute Difference (MAD) and Peak Signal-to-Noise Ratio (PSNR). PSNR is a variation of MSE, which is chosen as the

objective measure of video quality in this thesis. PSNR for luminance(Y), blue chrominance(C_b/U) and red chrominance(C_r/V) components are defined, respectively, as [7]

$$\text{PSNR}(Y) \triangleq 10\log_{10}\frac{255^2}{\frac{1}{N_Y}\sum_{i=1}^{N_Y}(p_{YO}[i]-p_{YR}[i])^2} \tag{1.3}$$

$$\text{PSNR}(U) \triangleq 10\log_{10}\frac{255^2}{\frac{1}{N_U}\sum_{i=1}^{N_U}(p_{UO}[i]-p_{UR}[i])^2} \tag{1.4}$$

$$\text{PSNR}(V) \triangleq 10\log_{10}\frac{255^2}{\frac{1}{N_V}\sum_{i=1}^{N_V}(p_{VO}[i]-p_{VR}[i])^2} \tag{1.5}$$

In Equation 1.3, 1.4, and 1.5, N_Y, N_U and N_V represent the number of luminance, blue chrominance and red chrominance pixels in a frame, respectively. $p_{YO}[i]$ and $p_{YR}[i]$ denote the luminance pixel value in an original and reconstructed frame, respectively. $p_{UO}[i]$, $p_{UR}[i]$, $p_{VO}[i]$ and $p_{VR}[i]$ are similarly defined for chrominance pixels. 255 is the highest-possible pixel value in a frame representing with 8 bits/pixel.

It should be noted that, comparing to RGB color space, the YC_bC_r (sometimes referred to as YUV) color space is a popular way of efficiently representing color images/videos, which makes use of the fact that the human visual system is less sensitive to color than luminance [7].

1.4 Organization of Thesis

This thesis is organized as follows:

Chapter 2 introduces necessary background information, such as the MPEG-4 standard, syntax hierarchy of MPEG-4 compressed bitstreams, video compressor/decompressor compliant with the MPEG-4 standard, materials related to turbo

codes and common error concealment techniques. This chapter also reviews the literatures on joint source and channel decoding for robust video/still image transmission.

Chapter 3 describes a simple joint source and channel decoding scheme, Syntax Based Error Concealment using Turbo Codes (SBECTC). Through SBECTC, some ideas of how to extract syntactic/semantic information from an MPEG-4 compressed bitstream and how to make use of the soft outputs from a turbo decoder to assist source decoding are introduced.

Chapter 4 presents the proposed scheme, Iterative Joint Source and Channel Decoding (IJSCD), which is based on the *iterative* feature of turbo codes. This chapter introduces a feedback scheme so that the extracted syntactic/semantic information can be available to the turbo decoder. Also, the convergence patterns of turbo decoding are checked in this chapter. At last, the performance of the proposed scheme, in terms of PSNR, BER, frame delay, overhead bits, picture quality and complexity, is investigated in Chapter 4, against two other schemes, the SBECTC and Turbo Decoding (TD).

Chapter 5 concludes this thesis and discusses the future work on iterative joint source and channel decoding using turbo codes.

Chapter 2

Background and Literature Review

This chapter presents the necessary background information on video compression/decompression, the MPEG-4 standard, turbo coding/decoding, error resilience and error concealment for video transmission, as well as joint source and channel decoding. In Section 2.1, a brief introduction on the MPEG-4 video compression standard is given. In particular, the hierarchy of the MPEG-4 compressed bitstream is described. Section 2.2 presents information on turbo codes, which are standard and powerful error control codes. The MAP decoding algorithm for turbo codes used in this thesis is also given in Section 2.2. In Section 2.3, a general introduction is presented on current error resilience and concealment schemes for compressed video. Section 2.4 reviews existing joint source and channel decoding schemes on video and image transmission. Finally, Section 2.5 concludes this chapter.

2.1 The MPEG-4 Standard

The Moving Pictures Experts Group (MPEG) is a committee formed under the Joint Technical Committee of the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC). It was formed in 1988. As a working group of ISO/IEC, MPEG is in charge of the development of international standards for compression, decompression, processing, and coded representation of moving pictures, audio and their combination [2]. In Sub-section 2.1.1, a brief review on the MPEG standards is presented. An overview on the MPEG-4 standard is also provided in this sub-section. The syntax of the MPEG-4 compressed bitstream is important in an environment where errors are inserted into the bitstream as in this thesis. Thus, the MPEG-4 syntax hierarchy is presented in Sub-section 2.1.2. Sub-section 2.1.3 and 2.1.4 describe a simple compressor and decompressor compliant with the MPEG-4 standard, respectively.

2.1.1 Context and Overview of the MPEG-4 Standard

So far, MPEG has produced MPEG-1 (ISO/IEC 11172), MPEG-2 (ISO/IEC 13818), MPEG-4 (ISO/IEC 14496) and MPEG-7 (ISO/IEC 15938), which have given rise to widely used commercial products and services. A brief introduction is given on each standard in the following.

MPEG-1 (ISO/IEC 11172) is the standard for storage and retrieval of moving pictures and associated audio on storage media, which was approved in November, 1992. MPEG-1 includes 5 parts: systems, video, audio, conformance testing and reference software. The video part provides efficient encoding of non-interlaced pictures with roughly Video Home System (VHS) quality at 1.15 Mb/s. Products such as Video-CD and MP3 are based on MPEG-1 [3, 8].

The subsequent MPEG-2 (ISO/IEC 13818) standard, on which such products as digital television set top boxes, Digital Versatile Discs (DVDs) and High-Definition TV (HDTV) are based, was approved in November 1994. MPEG-2 consists of all MPEG-1 coding tools, plus a considerable number of new ones including the tools for coding interlaced pictures and generating scalable bit streams. MPEG-2 has become very important because it has been chosen as the compression scheme for both the European Digital Video Broadcasting (DVB) and DVD. Since MPEG-2 supports a wide range of applications, it's divided into Profiles and Levels with a Profile describing a degree of complexity and a Level describing the resolution associated with the Profile [3, 8, 2].

MPEG-7 (ISO/IEC 15938) is the content representation standard for multimedia information search, filtering, management and processing, which was approved in July 2001 [8, 2].

MPEG-4 (ISO/IEC 14496) is an audiovisual coding standard that capitalized on advances in compression technology, after MPEG-1 and MPEG-2. The first set of MPEG-4 standard (so-called Version 1) was approved in October 1998. A major extension of the standard (so-called Version 2) was approved in December 1999.

Some work on extensions in specific domains is still in progress, which are MPEG-4 Version 3 (studio profile) and Version 4 (streaming profile) [8].

Initially, the objective of MPEG-4 was to address very low bitrate coding issues. However, its scope expanded to include all compression applications. It's known as the standard for multimedia applications that require powerful tools for interactivity. The most distinctive feature of MPEG-4 is that it enables the coding of individual objects. Different parts in a scene can be segmented, encoded and transmitted separately as visual objects and audio objects, which will be composited together at the MPEG-4 decoder. Visual objects in MPEG-4 can be of several different types, such as natural video, still texture, face, body and mesh objects [3, 9]. Moreover, the object based coding facilitates access to individual objects in a scene so that the interactivity can be reached at object level, not at frame level as MPEG-1/2 does. This kind of functionality is known as content-based interactivity [3, 8, 2, 9].

In object based coding, video objects need not to be of rectangular shape as in MPEG-1/2. That is, MPEG-4 supports arbitrarily shaped objects in contrast to previous block based schemes. (However, rectangular shaped objects are supported as a special case in MPEG-4.) Hence, when coding a video object, both texture data (corresponding to conventional video data but with a smaller size) and shape data have to be provided. Arithmetic encoding is used for shape coding. In order for higher coding efficiency, several new texture coding tools are employed such as AC/DC prediction for intra macroblocks, alternate scan modes, etc. [10]. Moreover, sprite, a special video object for background can substantially reduce the bits consumed in some cases such as football games since this kind of data is only sent once [2, 9].

MPEG-4 has also paid great attention to Error Resilience (ER), which makes it particularly suitable for use in error-prone environments [11, 9, 4]. There are four error resilient tools incorporated in MPEG-4 as listed in the following:

- Video Packet (VP) based resynchronization

  VP based resynchronization is similar to Slice in MPEG-1/2. However, the VP length in MPEG-4 is based on the number of bits contained in the VP, not on the number of macroblocks as MPEG-1/2 does. Thus, if the number of bits contained in the current VP exceeds a predetermined threshold, a new VP is created at the start of the next macroblock. The resynchronization marker is a specific code at the beginning of a VP and used to distinguish the start of a new VP.

- Header Extension Code (HEC)

  HEC is a single bit used to indicate whether additional information is available in the header of a VP. When it's turned on, information on time, motion vector and Video Object Plane (VOP) coding type are provided in the header so that each VP can be decoded independently even if the VOP header information is corrupted.

- Data partitioning

  Data partitioning is achieved by separating the motion and macroblock header information away from the texture information in a VP. Thus, this approach requires a second marker be inserted between motion and texture data. Its advantage is that, if texture data are corrupted, motion data can be used for error concealment.

- Reversible VLCs (RVLCs)

  RVLCs are designed such that the VLCs describing the texture information can be instantaneously decoded both in forward and backward directions in a VP with data partitioning. In this way, the number of corrupted bits can be reduced.

Note that all these four ER tools are implemented at VP layer. In this thesis, the only error resilient tool employed is VP resynchronization. More information about MPEG-4 error resilient tools are available from [11, 9, 4].

The targeted applications of MPEG-4 cover such areas as internet multimedia, interactive video games, interpersonal communications (videoconferencing, videophone etc.), interactive storage media (optical disks, etc.), multimedia mailing, networked database services (via ATM, etc.), remote emergency systems, remote video surveillance, wireless multimedia broadcasting applications [9]. Given the huge range of coding tools and the many applications, similar to MPEG-2, MPEG-4 is also subdivided into Profiles and Levels. For more detailed information about MPEG-4, please refer to [3, 2, 9, 12, 10].

2.1.2 MPEG-4 Video Hierarchy

One important principle in MPEG standards is that the encoder is not standardized. Only the bitstream the encoder produces and the procedure for decoding the bitstream are standardized. Any bitstream compliant with MPEG standards has to follow the syntax/semantics specified by the standards. Hence, some authors suggest that the syntax/semanics in MPEG standards can be used for error detection or error concealment [13, 1]. Thus, the syntax structure of MPEG-4 compressed bitstream is introduced in this sub-section [2, 9, 10]. Although MPEG-4 supports many different objects, the focus of this thesis is still on natural video. Figure 2.1 presents the complete hierarchy of an MPEG-4 natural video bitstream.

For natural video, the Video Sequence (VS) is the highest syntactic structure of the coded bitstream in MPEG-4, which is followed by one or more Video Objects (VOs) coded concurrently. The profile and level information are indicated in the VS header. In the VO header, objects' type, ID and video format such as NTSC, PAL are given [9].

In scalable coding, each VO can be coded into multiple Video Object Layers

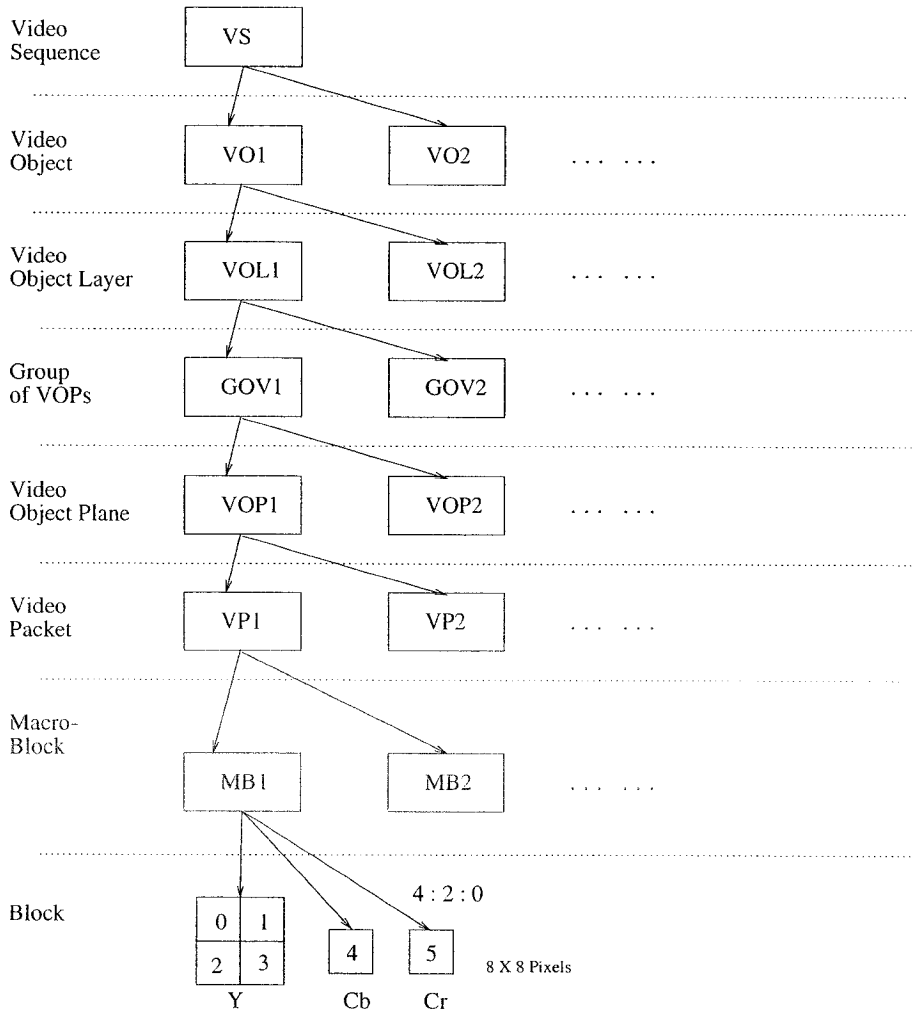

Figure 2.1: Hierarchical structure of MPEG-4 bitstream.

(VOLs), which consists of a base layer and several extensive layers. Each VOL corresponds to a certain spatial resolution of a VO. Its head information includes Layers' ID, aspect ratio, the horizonal and vertical size of the pixel aspect ratio, the chrominance format, the bitrate of the bitstream, the video buffering verifier size, the VOL shape, the VOP rate, the VOL width and height, the bits per pixel, the quantizer precision, the type and matrix for inverse DCT, the sampling factor in scalability, etc. If error resilient schemes are employed, this is also signalled in the VOL header. Note that MPEG-4 only supports 4:2:0 chrominance format [9].

Each time sample of a VO is defined as Video Object Plane (VOP), which is similar to a Frame in MPEG-1/2 if the VO consists of the entire frame. The Group of VOP (GOV) associates together several VOPs. The GOV header carries time information which is useful for random access and resynchronization in case of channel errors [9]. A typical GOV is illustrated in Figure 2.2 [9]. In this example, there are two B-VOPs between successive P-VOPs or successive I- and P-VOPs. Note that I- represents intra-coded VOPs, P- predictive-coded and B- bidirectionally predictive-coded VOPs. Particularly, an I-VOP is coded using information only from itself. A P-VOP is coded using motion compensated prediction from a past reference VOP. A B-VOP is coded using motion compensated prediction from a past and/or future reference VOP. The VOP header provides information such as the VOP coding type, time, the VOP width, height and spatial position, the quantizer value for de-quantizing the macroblock, etc. [9].

Each VOP is processed blockwise. Thus, below the VOP layer is the macroblock (MB). Each MB is a square of 16×16 pixels and comprises 4 luminance blocks (Y) and 2 spatially corresponding chrominance blocks (Cb or U and Cr or V) since the chrominance format is $4 : 2 : 0$ [9]. A block has 8×8 pixels. Note that motion estimation and compensation work on a MB basis and a block is the working unit for DCT/IDCT transform coding.

When the VP error resilience scheme is employed, the VP layer is inserted

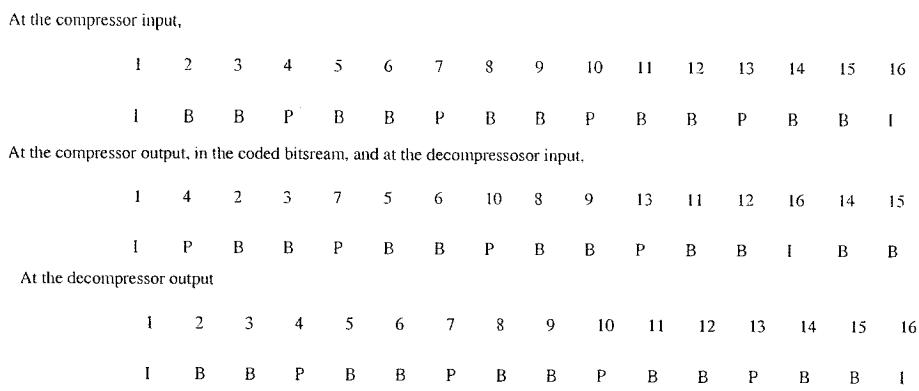

Figure 2.2: GOV structure and coding/decoding order.

between the VOP and MB layer. A VP contains several MBs depending on how many bits it can have. In the VP's header, the macroblock number in the current VP, and the quantizer value to be used for de-quantizing the MBs in the VP are provided [9].

2.1.3 Video Compressor

Although Sub-section 2.1.1 presented many powerful features of MPEG-4 such as object based coding/decoding, error resilience schemes, etc., this thesis only considers a simple video compressor and decompressor compliant with the MPEG-4 standard. Specifically, natural video is the input to the video compressor and the whole frame is treated as a video object so that video segmentation, shape coding and object composition at the decompressor side are not needed. Moreover, the compressor does not employ scalable coding. Note that in this thesis from this point on, the video source encoder/decoder is called video compressor/decompressor to distinguish it from the channel encoder/decoder.

As mentioned in the last sub-section, the MPEG standards don't standardize the compressor, but only specify the syntax of the compressed bitstream and the decoding process. Figure 2.3 shows the simplified MPEG-4 compressor [3, 10, 2] employed in this thesis. The functionality of each module in the figure is discussed

below.

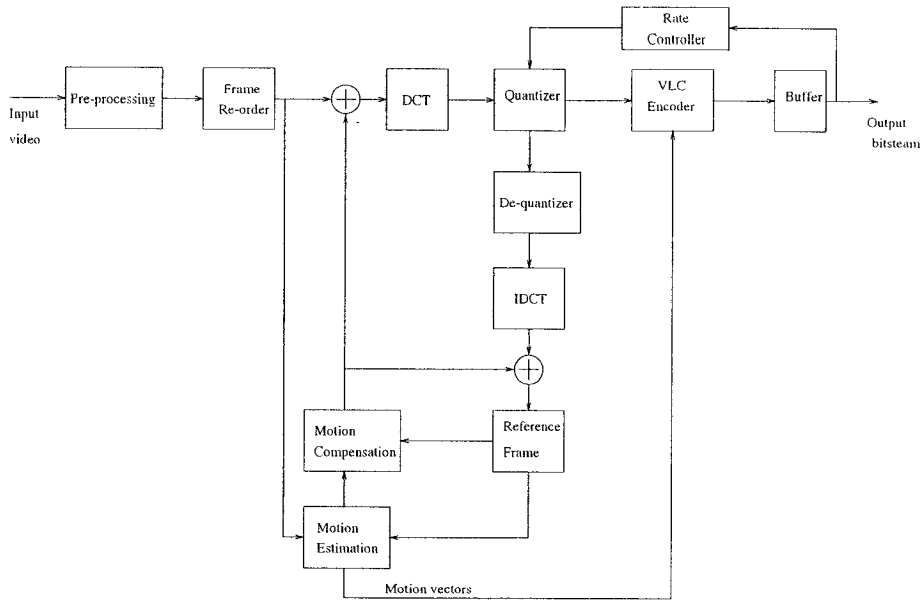

Figure 2.3: MPEG-4 compressor.

First, the input video is pre-processed, which is done by the Module Pre-processing. The preprocessing includes color system conversion, for example, from RGB to YCbCr, and sampling to achieve the suitable frame rate and frame size, etc. The pre-processed video are then re-ordered due to the presence of inter-coded frames(P- and B- frames). Figure 2.2 shows the coding order of a GOV, which is actually the GOV structure employed in the later simulations.

The Discrete Cosine Transform (DCT) reduces spacial redundancy in both intra-coded frames and residual frames, which are formed by subtracting the predictive frame from the frame being inter-coded. The Motion Estimation module compares the frame being inter-coded to a reference frame (or two reference frames for B-frame) to generate motion vectors, which indicate the location of the best matching macroblock in the reference frame. Based on the reference frame, Motion Compensation makes use of the motion vectors fed from the Motion Estimation to generate a predictive frame such that the temporal redundancy between frames is exploited. Note that motion compensation induces error propagation among frames,

as stated in Section 1.1.

The Quantizer in Figure 2.3 works on DCT coefficients for further compression. The serial concatenation of De-quantizer and Inverse DCT (IDCT) reconstructs the intra or residual frame as the case may be. The reconstructed residual frame adds the predictive frame to obtain the Reference frame in Figure 2.3. The quantized DCT coefficients are scanned in a specific order (so called zig-zag scanning) and then run-length coded using VLCs. Motion vectors are also coded using VLCs. These two jobs are done by the VLC Encoder. It's worth mentioning again that VLCs result in fragile bitstreams since any bit error in one VLC codeword can not only result in this codeword being un-decodable or wrongly decoded, but also render the following bits un-decodable or wrongly decoded.

The VLC coded bitstream goes into the Buffer before being sent out. According to the Buffer state, the Rate Controller in Figure 2.3 adjusts the step size of the Quantizer so as to keep the bitrate relatively constant as is commonly required.

2.1.4 Video Decompressor

The video decompressor [3, 10, 2] shown in Figure 2.4 reconstructs a video sequence according to the received compressed MPEG-4 bitstream generated by the video compressor described in the last sub-section. Correspondingly, this video decompressor is also a simplified MPEG-4 decompressor.

Quantized DCT coefficients and motion vectors come from decoding the corresponding VLCs. IDCT reconstructs the DCT coefficients for either an intra or residual frame depending on the frame coded type. For an inter-frame, Motion Compensation combines motion vectors and residual DCT coefficients to reconstruct it while only DCT coefficients are needed to reconstruct an intra-frame. The functionality of both the Buffer and the Frame Re-order are the same as in the compressor.

The decompressor, however, requires three important features because the

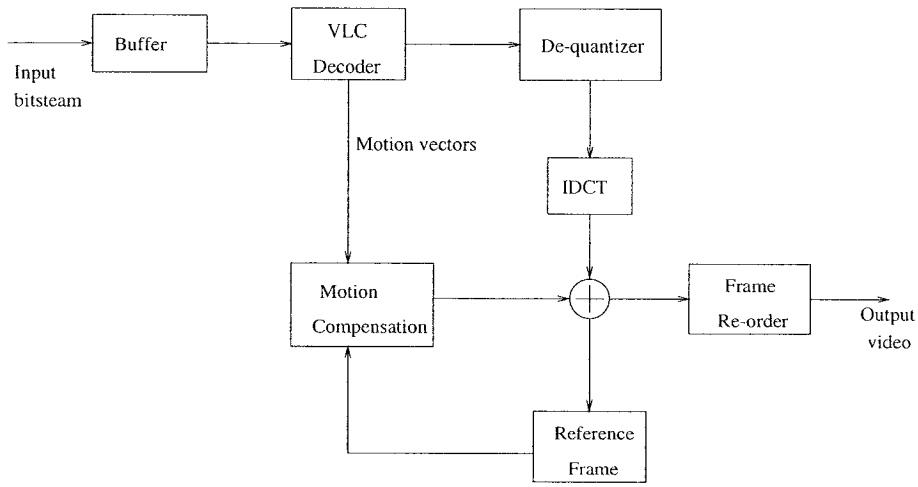

Figure 2.4: MPEG-4 decompressor.

received bitstream may contain channel induced errors: (1) It has the ability to recognize important syntax/semantic violations; (2) It must not stop decompressing when error bits are found; (3) The decompressor can output the bitstream which it has decompressed for bit error comparison and further processing.

2.2 Turbo Codes

Turbo code has been a popular research area since it was first proposed by Berrou *et. al.* in 1993 [14] for its excellent error performance. The block diagram of a conventional turbo coding/decoding process is shown in Figure 2.5. A conventional turbo encoder(please refer to Figure 2.5) includes two recursive systematic convolutional (RSC) encoders, which are parallel concatenated, an interleaver, and a puncturer. By interleaver, a turbo code has to work on a block-by-block basis and a turbo coding block usually is quite long in order for a desirable error performance. A turbo decoder consists of two maximum *a posteriori* (MAP) decoders. The two MAP decoders shown in Figure 2.5 are linked through the same interleaver as used by the turbo encoder and the de-interleaver, which corresponds to the interleaver. Thus, soft decisions made by the two MAP decoders can be iteratively passed from the

output of one decoder to the input of the other decoder. If puncturing is used in the encoder part, the corresponding de-puncturer is also required in the decoding part. It can be seen from the above description that such functions as parallel concatenation, large block length, interleaving, iteration and soft decoding are included in turbo codes, which give rise to their excellent error performance [15].

2.2.1 Shannon Limit and Turbo Code Error Performance

According to Shannon's Theorem, the minimum required channel SNR $\frac{E_b}{N_0}$ for error-free transmission is -1.59 dB [5]. Note that, as the Shannon limit of -1.59 dB is approached, the required system bandwidth approaches infinity. Therefore, the Shannon limit represents a theoretical bound, but it is not a practical goal.

In the case of Binary Phase Shift Keying (BPSK) modulation, several authors use BER $P_b = 10^{-5}$ and $\frac{E_b}{N_0} = 0.2$ dB as a *pragmatic* Shannon limit for a rate 1/2 code [5].

Berrou *et. al.* in [16] employ a turbo code with generator matrix

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

and a nonuniform interleaver with $N = 256 \times 256 = 65,536$ bits. After 18 decoding iterations, the BER is less than 10^{-5} at $\frac{E_b}{N_0} = 0.7$ dB using a rate $\frac{1}{2}$ code over an Additive White Gaussian Noise (AWGN) channel. Thus, the error performance of this turbo code is only 0.5 dB away from the pragmatic Shannon limit and that is why turbo code is so attractive.

2.2.2 Turbo Encoder

A conventional turbo encoder in Figure 2.5 is formed by parallel concatenation of two RSC encoders linked through an interleaver. In general, the two RSC encoders

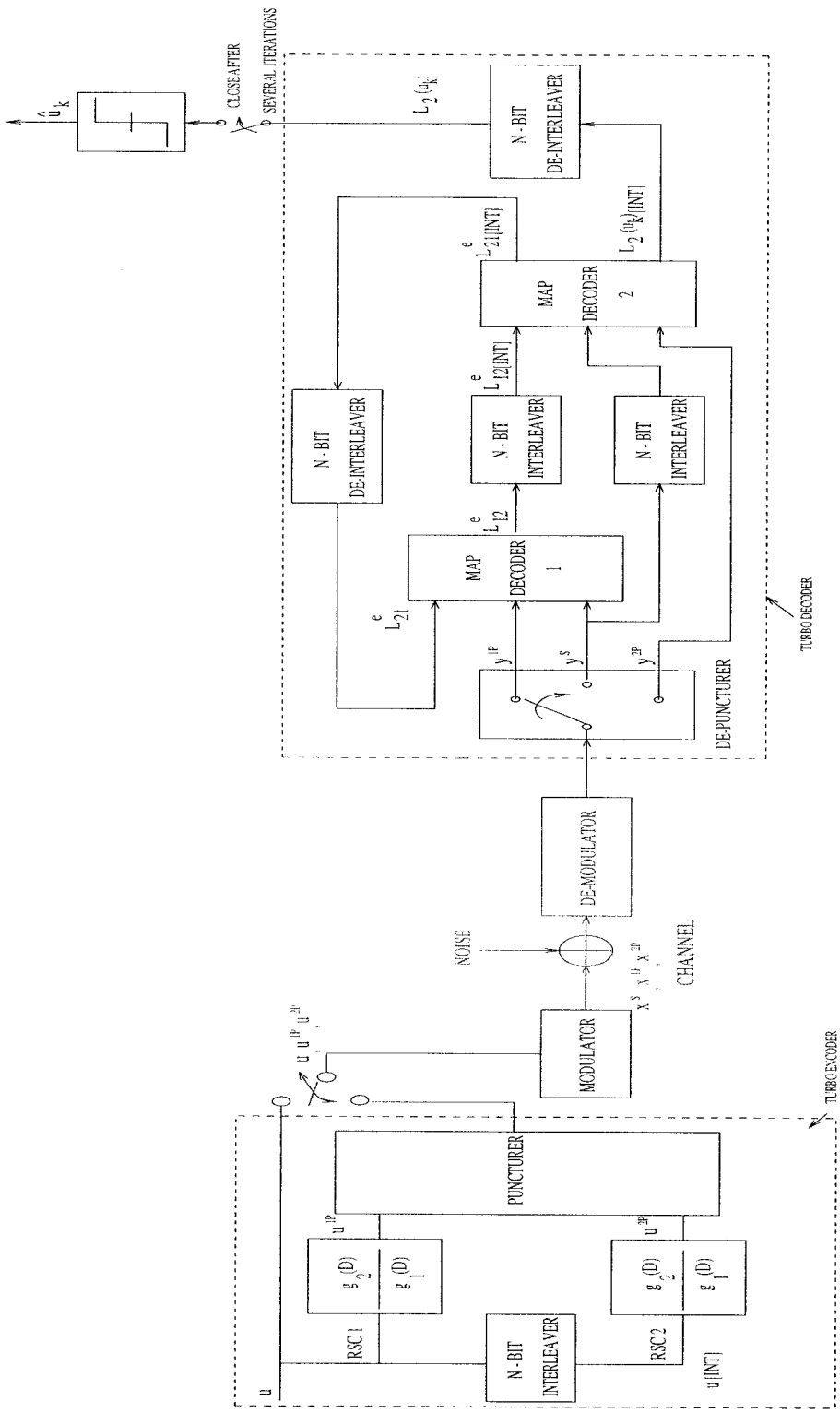

Figure 2.5: Block diagram of turbo coding/decoding

are identical. In some cases, a turbo encoder also includes a puncturer to increase its code rate.

In the turbo coding process, the input information bits feed into the first RSC encoder and, after having been interleaved by the interleaver, enter into the second encoder. The codeword of a turbo code consists of the input bits to the first RSC encoder and the parity check bits generated by both encoders. The parity bits can be punctured according to the available bandwidth or power.

Compared with nonsystematic convolutional code (NSC), the RSC code is chosen in a turbo encoder because of the following reason. Whereas the generator matrix of a rate 1/2 NSC code has the form $G_{NSC} = [g_1(D) \quad g_2(D)]$, the equivalent RSC encoder has the generator matrix as $G_{RSC} = [1 \quad \frac{g_2(D)}{g_1(D)}]$. Comparing the two generator matrices, it's easy to see that both RSC and NSC encoders have the same set of codewords. However, the input-output weight correspondence are different between NSC and RSC encoders. Especially, the weight-1 input sequences to an NSC encoder can generate finite output codewords. In contrast to an NSC encoder, the weight-1 input sequences to an RSC encoder can only generate infinite output codewords if not considering the turbo coding block length. Moreover, for NSC codes, error paths with weight-1 input sequences are dominant on the turbo code error performance and in this case, the BER is independent of the interleaver length N [15, 17]. Thus, no performance gain can be achieved by interleaving for NSC codes. That's why turbo codes do require RSC encoders to work effectively.

2.2.3 Interleaver/De-interleaver

Interleaving is a process of reordering an input bit sequence. The inverse of this process is called de-interleaving which restores the received sequence into their original order.

The basic role of an interleaver is to construct a long block code from small memory RSC codes since long codes provides the possibility of approaching the

Shannon limit. Another role of an interleaver is to de-correlate the input sequences to the two constituent MAP decoders. In this case, after one decoder corrects some of the errors, some of the remaining errors should be correctable in the other decoder. The final role of the interleaver is to break low weight input sequences. When the parity check sequence out of the first RSC encoder is of low weight for a specific input sequence, a well-designed interleaver can generate a high weight parity check sequence out of the second RSC encoder so that the overall weight of the code can be increased.

The length and structure of an interleaver can considerably affect the turbo code error performance. At low channel SNRs, the interleaver length is a dominant factor. However, both the interleaver length and structure play an important role in determining the code performance at high channel SNRs [18].

Commonly used interleaving techniques include block interleavers, convolutional interleavers, random interleavers and code matched interleavers. In this thesis, a random interleaver is employed. In random interleaving, a block of N input bits is read into the interleaver and read out randomly.

2.2.4 Turbo Decoder and MAP Decoding Algorithm

The turbo decoder in Figure 2.5 consists of two constituent MAP decoders which are serially concatenated through two interleavers and a de-interleaver. The interleavers are identical to the ones used in the turbo encoder and the de-interleaver is the reverse of the interleaver.

During the turbo decoding process, first, MAP decoder 1 takes as input the received information sequence $\mathbf{y}^s$ and the received parity check sequence $\mathbf{y}^{1p}$, which is generated by the RSC encoder 1. The MAP decoder 1 then produces a soft output, which is interleaved and used as *a priori* probabilities of the information sequence for the MAP decoder 2. The other two inputs to the MAP decoder 2 are the interleaved received information sequence and the received parity check sequence $\mathbf{y}^{2p}$, which is

generated by the RSC encoder 2. Similarly, the MAP decoder 2 also generates a soft output, which is de-interleaved and used as *a priori* probabilities of the information sequence for the MAP decoder 1. This kind of work between two MAP decoders can be repeated several times according to the performance requirement. Finally, after several iterations, the turbo decoder makes hard decision on the information sequence.

The decoding algorithm employed is the so called MAP decoding algorithm and that's why the two constituent decoders are named MAP decoders. Berrou *et al.* [14] in the first paper about turbo codes utilized and modified the symbol-by-symbol maximum *a posterior* (MAP) decoding algorithm of Bahl *et al.* (usually referred as BCJR algorithm) for convolutional codes [19]. The MAP decoding algorithm for RSC codes and turbo codes assuming an AWGN channel and BPSK modulation described in the following comes from [20] and [21]. These two papers give some detailed derivations and information of MAP decoding. Before further illustrating the MAP algorithm, the following notations should be given at first:

- N is the interleaver length.
- m is the RSC encoder's memory.
- k refers to a time instant.
- S is the set of all 2^m states of the RSC encoder.
- s_k is the state of the encoder at time k.
- $\mathbf{u} = (u_1,\ u_2,\ \cdots,\ u_N)$ is the input information bit sequence.
- $\mathbf{x}^s = (x_1^s,\ x_2^s,\ \cdots,\ x_N^s)$ is the modulated information bit sequence.
- $\mathbf{u}^p = (u_1^p,\ u_2^p,\ \cdots,\ u_N^p)$ is the parity check bit sequence.

- $\mathbf{x}^p = (x_1^p,\ x_2^p,\ \cdots,\ x_N^p)$ is the modulated parity check bit sequence. $\mathbf{x}^{1p}$ and $\mathbf{x}^{2p}$ represent the modulated parity check bit sequence generated by the RSC encoder 1 and 2, respectively.

- $\mathbf{y} = (y_1,\ y_2,\ \cdots,\ y_N)$ is the received noisy bit sequence.

- $y_k = (y_k^s,\ y_k^p)$ is the noisy version of $(x_k^s,\ x_k^p)$. $\mathbf{y}^{1p}$ and $\mathbf{y}^{2p}$ corresponds to the noisy version of $\mathbf{x}^{1p}$ and $\mathbf{x}^{2p}$, respectively.

- $\mathbf{y}_i^j = (y_i,\ y_{i+1},\ \cdots,\ y_j)$.

The main idea of BCJR Algorithm follows from the Markov property, that is, if s_k is known, bits after time k don't depend on $\mathbf{y}_1^k$. The detailed BCJR algorithm for decoding convolutional codes and linear block codes can be found in [19].

For a single MAP decoder employing BCJR algorithm, the MAP decoder decides $u_k = 1$ if $P(u_k = 1|\mathbf{y}) > P(u_k = 0|\mathbf{y})$ and $u_k = 0$ otherwise. The decision $\hat{u}_k$ is given in terms of natural logarithm as

$$\hat{u}_k = \mathbf{sign}[L(u_k)] \tag{2.1}$$

where $L(u_k)$ is the logarithm *a posteriori* probability (LAPP) ratio and defined as

$$L(u_k) \triangleq \log\left(\frac{P(u_k = 1|\mathbf{y})}{P(u_k = 0|\mathbf{y})}\right) \tag{2.2}$$

Incorporating the RSC code's trellis and the Markov property presented above, Equation 2.2 may be rewritten as

$$L\left(u_k\right) = \log\left(\frac{\sum_{S^+} \alpha_{k-1}(s') \cdot \gamma_k\left(s', s\right) \cdot \beta_k\left(s\right)}{\sum_{S^-} \alpha_{k-1}(s') \cdot \gamma_k\left(s', s\right) \cdot \beta_k\left(s\right)}\right) \tag{2.3}$$

where S^+ is the set of order pairs (s', s) corresponding to all state transitions from $s_{k-1} = s'$ to $s_k = s$ caused by input bit $u_k = 1$, and S^- is similarly defined for $u_k = 0$.

In Equation 2.3, $\alpha_k\left(s\right)$ and $\beta_k\left(s\right)$, called as *forward state metric* and *backward state metric* respectively, are computed recursively as

$$\alpha_k(s) = \frac{\sum_{s'} \alpha_{k-1}(s')\gamma_k(s',s)}{\sum_s \sum_{s'} \alpha_{k-1}(s')\gamma_k(s',s)}. \tag{2.4}$$

$$\beta_{k-1}(s') = \frac{\sum_{s} \beta_k(s)\gamma_k(s',s)}{\sum_s \sum_{s'} \alpha_{k-1}(s')\gamma_k(s',s)}. \tag{2.5}$$

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

$$\gamma_k(s',s) \triangleq p(s_k = s, y_k | s_{k-1} = s'). \tag{2.6}$$

The computation of the probabilities $\gamma_k(s',s)$ will be given later.

However, the BCJR algorithm only applies to a single decoder. In order to take full advantage of a bit's *a priori* information available from the previous MAP decoder, some modification on BCJR algorithm has to be done to make possible iterative exchanging information between two MAP decoders, which is the so-called turbo decoding. The *a priori* information of each bit generated by the previous MAP decoder $L^e(u_k)$ is defined by

$$L^e(u_k) \triangleq \log\left(\frac{P(u_k = 1)}{P(u_k = 0)}\right) \tag{2.7}$$

The branch metric $\gamma_k(s',s)$ is then computed as

$$\gamma_k(s',s) = \exp\left[\frac{1}{2}x_k\left(L^e(u_k) + L_c y_k^s\right)\right] \cdot \gamma_k^e(s',s) \tag{2.8}$$

where

$$L_c \triangleq \frac{4E_c}{N_0} \tag{2.9}$$

and

$$\gamma_k^e(s',s) \triangleq exp[\frac{1}{2}L_c y_k^p x_k^p]. \tag{2.10}$$

In Equation 2.9, $E_c = rE_b$ with E_b and r representing the bit energy and the code rate, respectively. N_0 is the one sided noise power spectral density.

After incorporating the *a priori* information $L^e(u_k)$, the LAPP ratio in Equation 2.3 can be further represented as

$$L(u_k) = \underbrace{L_c y_k^s}_{\substack{\text{channel} \\ \text{value}}} + \underbrace{L^e(u_k)}_{\substack{a\ priori \\ \text{information}}} + \underbrace{\log\left(\frac{\sum_{S+}\alpha_{k-1}(s')\cdot\gamma_k^e(s',s)\cdot\beta_k(s)}{\sum_{S-}\alpha_{k-1}(s')\cdot\gamma_k^e(s',s)\cdot\beta_k(s)}\right)}_{\substack{\text{extrinsic} \\ \text{information}}} \quad (2.11)$$

The first term $L_c y_k^s$ in the right hand side of Equation 2.11 is sometimes called *channel value* representing the information from channel. The second term $L^e(u_k)$ represents the *a priori* information about u_k provided by the previous MAP decoder, and the third term represents the *extrinsic information* only available in the current decoder, which can be passed on to the subsequent decoder as *a priori* information. In particular, Equation 2.11 can be rewritten as

$$L_1(u_k) = L_c y_k^s + L_{21}^e(u_k) + L_{12}^e(u_k) \quad (2.12)$$

$$L_2(u_k)_{[INT]} = L_c y_{k[INT]}^s + L_{12}^e(u_k)_{[INT]} + L_{21}^e(u_k)_{[INT]} \quad (2.13)$$

for MAP decoder 1 and 2, respectively, where "[INT]" represents the interleaving operation. $L_{ij}^e(u_k)$ $(i,j = 1,2)$ is the third term in the right hand side of Equation 2.11. After the final iteration, MAP decoder 2 makes a decision on each u_k. If $L_2(u_k) > 0$, decode u_k as 1; otherwise, as 0.

Equation 2.12 and 2.13 clearly shows that the output from the MAP decoder takes soft value. The reliability information about each decoded information bit is the absolute value of $L_i(u_k)$. Generally, correctly decoded bits most likely have high $|L_i(u_k)|$ values while wrongly decoded bits have small $|L_i(u_k)|$ values which are close to the threshold 0 as illustrated in Figure 2.6 [22].

The initial conditions in the MAP decoding algorithm are listed as follows:

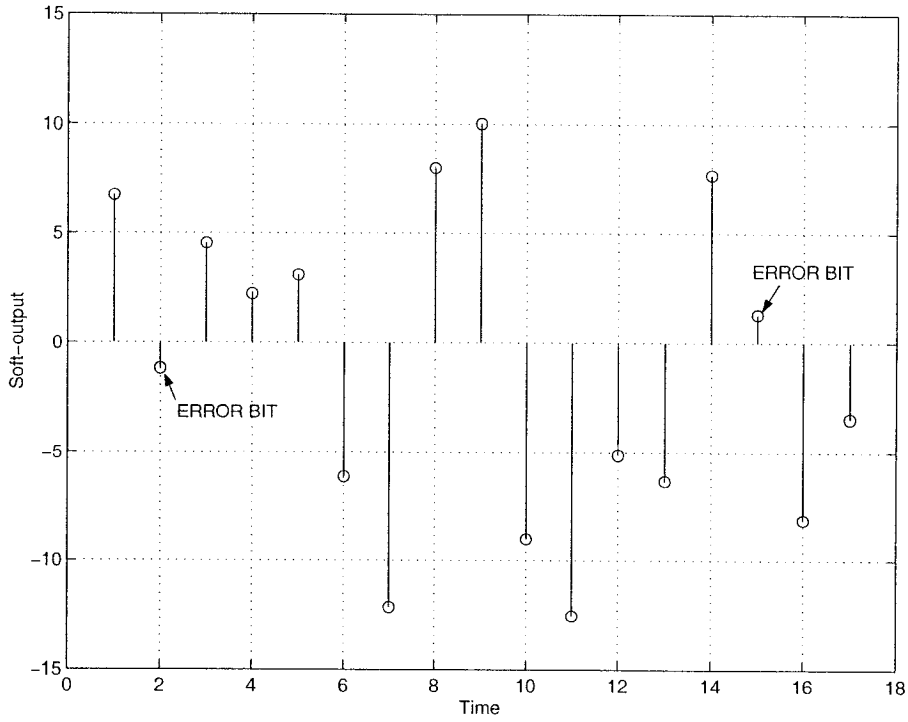

Figure 2.6: Illustration of the soft outputs from a turbo decoder

For MAP decoder 1,

$$\alpha_0^{(1)}(s) = \begin{cases} 1 & \text{if } s = 0 \\ 0 & \text{if } s \neq 0 \end{cases} \tag{2.14}$$

$$\beta_N^{(1)}(s) = \begin{cases} 1 & \text{if } s = 0 \\ 0 & \text{if } s \neq 0 \end{cases} \tag{2.15}$$

$$L_{21}^{e}(u_k) = 0 \text{ for } k = 1, 2, \ldots, N. \tag{2.16}$$

For MAP decoder 2,

$$\alpha_0^{(2)}(s) = \begin{cases} 1 & \text{if } s = 0 \\ 0 & \text{if } s \neq 0 \end{cases} \tag{2.17}$$

$$\beta_N^{(2)}(s) = \frac{1}{2^m} \text{ for all s.} \tag{2.18}$$

$L_{12}^{e}(u_k)_{[INT]}$ is the output from the MAP decoder 1 after the first half iteration.

For more detailed derivation of the MAP algorithm, please refer to [19, 18, 20, 21].

2.2.5 Complexity of MAP Decoding Algorithm

For a $(n, 1, m)$ RSC code, the complexity of MAP decoding algorithm per time unit is illustrated as below

1. Addition: $7 \cdot 2^m - 1$.
2. Multiplication: $5 \cdot 2^{m+1} + 4$.
3. Exponential: 2^{m+2}.

m is the memory of the RSC code.

The total complexity can be estimated by multiplying the time-unit complexity listed above by the interleaver length and iteration times.

2.3 Error Resilience and Error Concealment for Video Communication

As presented in Chapter 1, an unavoidable problem with video communication is that the transmitted video bitstream usually suffers from channel noise. In general, the channel induced errors can be classified into three categories: random bit errors caused by imperfections of physical channels, burst errors in storage media or in wireless networks resulted from multipath propagation, and packet loss caused by network congestion in packet networks [4, 1]. Moreover, a compressed video bitstream is extremely sensitive to errors due to the use of predictive coding and VLC codes as described in Section 1.1 and 2.1.3. Hence, many techniques have been developed to combat these channel induced errors. These techniques are commonly

divided into three types called Forward Error Correction (FEC), Error Resilience (ER) and Error Concealment (EC) [23, 1]. These three types of techniques for robust video communication are further described in the following.

In FEC, a video bitstream is protected using channel error control codes by systematical inserting extra redundancy bits. So far, Unequal Error Protection (UEP) schemes are the most popular among FEC schemes for video transmission, in which different error protection is provided based on the importance of the bits, i.e., more important bits are protected with stronger channel codes [23, 1].

ER schemes mainly refer to what a video compressor employs to make a compressed bitstream robust to channel errors [24]. MPEG-4 supports the ER schemes including VP based resynchronization, HEC, data partitioning and RVLCs, which have been discussed in Section 2.1.1.

Although video compression schemes can heavily reduce the temporal and spatial redundancies in a natural video, some redundancies remain. EC schemes used in a video decompressor make use of residual redundancy to minimize the artifacts caused by transmission errors. These EC schemes are subdivided into four categories: simple concealment, temporal concealment, spatial concealment and frequency concealment according to the redundancy they use [23, 1]. Lynch *et al.* [13] introduces an EC method based on the syntax structure of compressed bitstreams. In [25], a hybrid spatial/temporal EC approach is presented for MPEG-4 video transmission. However, error detection has to be done before using any EC scheme. Common ways for error detection includes using FEC codes, characteristics of natural videos and the syntax structure of a compressed bitstream [1, 24].

Clearly, the three techniques for robust video transmission introduced above are not mutually exclusive, but can be used jointly. In fact, most papers in this field present joint schemes and many references are available from [1, 24].

The proposed robust video transmission scheme in this thesis is also a joint scheme, which combines an FEC code (turbo codes), an ER scheme (VP based

resynchronization) and syntax based error concealment together. Moreover, the proposed scheme provides for interactivity between the channel decoder and the source decoder, from which the syntax based EC scheme can benefit.

2.4 Literature Review on Joint Source and Channel Decoding for Image/Video Transmission

Some common techniques for robust video transmission have been introduced in the last section such as FEC coding, error resilience and concealment schemes. If FEC is employed for error protection, many authors [26, 13, 27, 28, 29, 30, 22] have pointed out that the redundancy left in a source after compression can be utilized jointly by a channel decoder while a source decoder uses the information from the channel decoder to assist error concealment, which is called Joint Source and Channel Decoding (JSCD) and the focus of this thesis. Thus, this section will review some papers in this area. Special attention is paid to the joint schemes using turbo codes or convolutional codes for image/video transmission.

In [29], a joint source and channel decoding approach for image transmission over AWGN channel is described. The channel code is a convolutional code and *a priori* Soft Output Viterbi Algorithm (APRI-SOVA) is used for decoding it. The source coders employ the nonadaptive DCT coding and subband coding with Lloyd-Max quantization. The quantized coefficients are coded using Fixed Length Coding (FLC) and DC coefficients are not predictively coded to reserve the bit correlation among neighboring samples. Then, the reserved redundancy in coded bits is exploited in that the final soft output for each bit from the channel decoder linearly combines the soft output of the current bit and the value linearly predicted from those of previously decoded neighboring bits. However, this method is not formulated using the standard method of compressing quantized non-DC coefficients and predictive coding for DC coefficients with VLC's.

Kliewer *et al.* in [30] introduce an iterative decoding scheme for subband coded image transmitted over an AWGN channel. A soft-input *a posteriori* probability (APP) source decoder is developed based on exploiting implicit residual redundancy in the index vectors after vector-quantization. The channel code is also a convolutional code and MAP decoding algorithm [20] is used for channel decoding. Then, the source decoder and channel decoder work together like turbo decoding of two serially concatenated codes. Note that subband coding is not the standard method for image compression.

An iterative joint source and channel decoding scheme using turbo codes is found in [22] for MPEG-1 video and JPEG image transmission. This scheme has also been used for vector quantized image [31] and subband coded image [32]. The proposed scheme includes error-free source information feedback, error-detected source information feedback, and the use of channel soft outputs for source post-processing. The feedback scheme is a weight operation on the extrinsic information passed between the two constituent decoders inside a turbo decoder based on the extracted source information, which varies along with different visual sources and source compression methods. The work on MPEG-1 video in [22] only makes use of the high degree of predictability of the MPEG-1 start codes. Then, error-free source information feedback scheme is employed for these start codes. Other than these start codes it does not appear to use other syntax/semantic information of the compressed bitstream.

Aign in [33] studies error concealment techniques for MPEG-2 inter coded pictures with the assistance of the soft output values from a channel decoder. In [33], a RS (204, 188, 17) code is employed as outer channel code and a convolutional code as inner code, which is decoded using the Soft Output Viterbi Algorithm (SOVA). The soft output from the SOVA decoder are applied to detect errors where the RS code may fail.

Bystrom *et al.* [34] considers the problem of soft decoding of VLCs and the

proposed method is applied to MPEG-4 videos. However, the statistics of the employed VLC are need to be investigated at first and the number of bits and VLC codes used in the coded bitstream should be sent as side information and received correctly.

2.5 Summary

This chapter has provided some background information related to the work in this thesis. The background information includes the introduction on the MPEG-4 standard and compressor/decompressor compliant with the MPEG-4 standard. Particularly, the syntax structure of an MPEG-4 compressed bitstream has been described in this chapter. The introduction on turbo codes and the MAP decoding algorithm for turbo codes has also been given in this chapter. Since the proposed joint source and channel decoding scheme will employ some error resilience and concealment techniques, some information on error resilience and concealment techniques is provided in this chapter. Finally, this chapter has reviewed the literatures on joint source and channel decoding schemes for video/image transmission.

Chapter 3

Syntax Based Error Concealment (SBEC) Using Turbo Codes

The syntax structure of an MPEG-4 bitstream, the reliability associated with the soft outputs from a turbo decoder and common EC techniques are discussed in Chapter 2. These important concepts form the basis of Syntax Based Error Concealment using Turbo Codes (SBECTC), which is the focus of the current chapter. In this chapter, first, the residual information in an MPEG-4 bitstream, which can be used for detecting syntactic/semantic errors in the MPEG-4 bitstream, is given in Section 3.1. Section 3.2 reviews previous work on using residual information in an MPEG bitstream for error concealment. Then, Section 3.3 presents the detailed information on the SBECTC. Finally, simulation results of the SBECTC are given in Section 3.4.

3.1 Residual Source Information in an MPEG-4 Bitstream

As discussed in Chapter 2, motion compensation/estimation and the DCT employed by MPEG-4 standard [9] can dramatically reduce the spatial and temporal redundancy in a video. However, some redundancy is still available after compression, which have been widely used for spatial, temporal or spatial-temporal error concealment. Especially, the syntax and semantics specified by MPEG-4 standard [9] are residual information inhered in an MPEG-4 coded bitstream and can be applied for error concealment [13].

An MPEG-4 bitstream is structured into several layers corresponding to different objects, locations and time instants. From top to bottom, these layers are: Video Sequence (VS), Video Object (VO), Video Object Layer (VOL), Group of VOPs (GOV), Video Object Plane (VOP), Video Packet (VP), macroblock (MB) and block [2, 9, 10]. Each layer includes certain components and these components are organized in a specific order. All the layers above macroblock layer include a special component, start-code or resynchronization marker. The bits spent on each

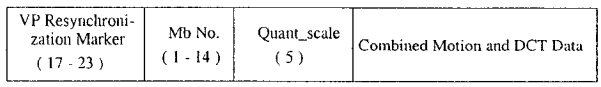

Figure 3.1: Bitstream organization within an MPEG-4 video packet. Numbers represents the bit length of the corresponding component.

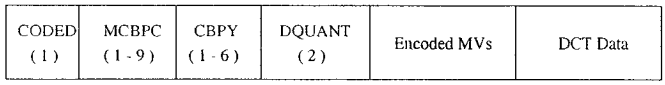

Figure 3.2: Bitstream organization within an MPEG-4 macroblock. Numbers represents the bit length of the corresponding component.

component have specific length and meaning, and are allotted according to various code tables in MPEG-4 standard [9]. Figure 3.1 and Figure 3.2 show the bitstream components, their order and corresponding length in a VP and a MB, respectively. This syntax, on one hand, ensures that the MPEG-4 standard is generic so that it can serve a wide range of applications. On the other hand, it can be used as a powerful tool for error concealment since the violation of MPEG-4 syntax/semantics can be easily detected by an MPEG-4 decompressor and the syntactic/semantic errors often render serious video quality degradation.

The examples of syntactic/semantic violation are listed below:

- The motion vectors are out of range.
- Start codes are not at the beginning of the corresponding layer.
- The decoded bits can not be found at any code table in MPEG-4 standard.
- The number of macroblocks doesn't match for the number specified in a video packet header.
- The DCT coefficient is out of range.
- The number of DCT coefficients of a block exceeds 64.

In addition to syntactic/semantic residual information in a compressed video bitstream, spatial smoothness [35] is another kind of residual information available

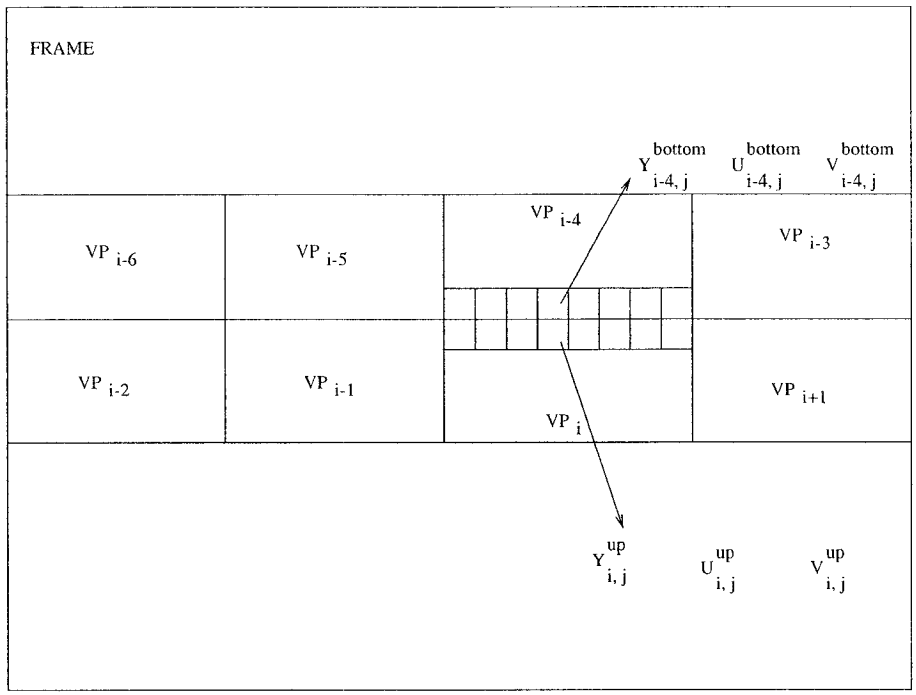

Figure 3.3: Illustration of pixels used in discontinuity measure.

from a partially decompressed video because abrupt discontinuity is very uncommon in a natural scene due to the video capture process. Considering 4 VPs per row, the discontinuity measure for luminance component M_Y is defined as [35]

$$M_Y \triangleq \frac{1}{N_Y} \sum_{j=0}^{N_Y-1} \left(Y_{i-4,j}^{\text{bottom}} - Y_{i,j}^{\text{top}} \right) \tag{3.1}$$

where $Y_{i,j}^{\text{top}}$ is the luminance pixel value in the top row of VP_i while $Y_{i-4,j}^{\text{bottom}}$ represents pixel value in the bottom row of the VP above VP_i. N_Y is the number of pixels in a line of VP_i.

Similarly, the discontinuity measure for chrominance components M_U and M_V are defined as

$$M_U \triangleq \frac{1}{N_U} \sum_{j=0}^{N_U-1} \left(U_{i-4,j}^{\text{bottom}} - U_{i,j}^{\text{top}} \right) \tag{3.2}$$

$$M_V \triangleq \frac{1}{N_V} \sum_{j=0}^{N_V-1} \left(V_{i-4,j}^{\text{bottom}} - V_{i,j}^{\text{top}} \right) \tag{3.3}$$

where $U_{i,j}^{\text{top}}$ and $V_{i,j}^{\text{top}}$ are the chrominance pixel values in the top row of VP_i. $U_{i-4,j}^{\text{bottom}}$ and $V_{i-4,j}^{\text{bottom}}$ are pixel value in the bottom row of the VP above VP_i. N_U and N_V are the number of chrominance pixels in a row of VP_i.

The overall discontinuity measure is defined as

$$M_A \triangleq \left(4M_Y^2 + M_U^2 + M_V^2\right)^{\frac{1}{2}} \tag{3.4}$$

for a video with format $4:2:0$.

For error concealment, the smallest M_A is preferred since it means the best spatial smoothness, when considering several candidates for a VP. Figure 3.3 illustrates the pixels used in discontinuity measure.

3.2 Review of Previous Work

Previous work on using residual information in an MPEG-2 bitstream to combat channel noise employed different channel error control codes and different schemes for generating slice candidates. These schemes are further illustrated in the following.

1. Syntax based error concealment (SBEC) [13]

   In [13], an MPEG-2 bitstream is encoded using a simple parity check code, which adds 1 parity bit for each error detection (ED) block of 12 bits. At the receiver end, corrupted ED blocks are located by parity checking. If an ED block is flagged as corrupted, it's known that one of the 13 bits is erroneous. Hence, the slice containing this ED block is decoded up to 13 times and each time a different bit in this ED block is toggled. Accordingly, two corrupted

ED blocks in a slice will result in 169 decodings, etc. The first slice candidate without syntax violation during the toggling process will be accepted as the correct one. Clearly, SBEC uses only the syntactic/semantic residual information in an MPEG-2 bitstream for error concealment.

2. Syntax and discontinuity based error concealment (SDBEC) [35]

   SDBEC is an extension of SBEC, which not only makes use of syntactic/semantic residual information, but also uses discontinuity measure to assist error concealment. The channel code employed in SDBEC is the same as in SBEC. SDBEC first works like SBEC to check if a slice candidate has any syntax error, but SDBEC keeps all the slice candidates without syntax violation. Then, the slice candidate with the smallest M_A is chosen as the best slice candidate.

3. Joint forward error correction and error concealment for compressed video [28]

   Mei *et al.* [28] works on MPEG-2 videos, too. A $(16, 8)$ quasi-cyclic code with minimum Hamming distance 5 is employed for forward error correction (FEC). In general, the classic FEC approach generates only one slice candidate which has the shortest Hamming distance to the received bits. If more than one slice candidate have the shortest Hamming distance, a random one among them is chosen. The scheme described in [28] not only preserves all the candidates with the shortest Hamming distance, but also includes the candidates which have the shortest Hamming distance plus 1 or plus 2, which is the way how slice candidates are constructed. Considering of choosing the best slice candidate, Mei *et al.* in [28] develop a measure which combines channel information (Hamming distance), syntax information and discontinuity measure together. In this scheme, the slice candidates from the shortest Hamming distance group have the highest priority and the ones from the shortest Hamming distance plus 2 group have the least priority of being chosen as the best candidate. The

different weights on the priority are decided experimentally.

4. Syntax and discontinuity based error concealment for compressed video in a packet environment [27]

   Lynch *et al.* [27] work on MPEG-2 videos and aims to apply SDBEC in a packet environment. In [27], the errors considered are lost packets which are generally caused by network congestion in a packet switched network and no channel coding is employed. The bits from a single slice are distributed among several packets so that the loss of a packet will result in only one or a few errors in one slice. The loss of a packet is detected by the de-packetizer at the receiver side and the lost bits are treated as erasures. Syntax information and discontinuity measure are used to find the best slice candidates for a corrupted slice.

3.3 Syntax Based Error Concealment Using Turbo Codes

Syntax Based Error Concealment using Turbo Codes (SBECTC) follows the previous work in utilizing the source syntax/semantics for error detection and concealment. The turbo code employed in the SBECTC is introduced in Sub-section 3.3.1. The channel condition and modulation/de-modulation scheme are described in Sub-section 3.3.2.

Figure 3.4 shows the block diagram of the SBECTC. The Video Compressor and Decompressor have been introduced in Sub-section 2.1.3 and 2.1.4, respectively. The turbo decoder and the employed MAP decoding algorithm have been discussed in Section 2.2.

Since the current work uses turbo codes, the soft information of each bit is available from the turbo decoder. To utilize this information, the relation between

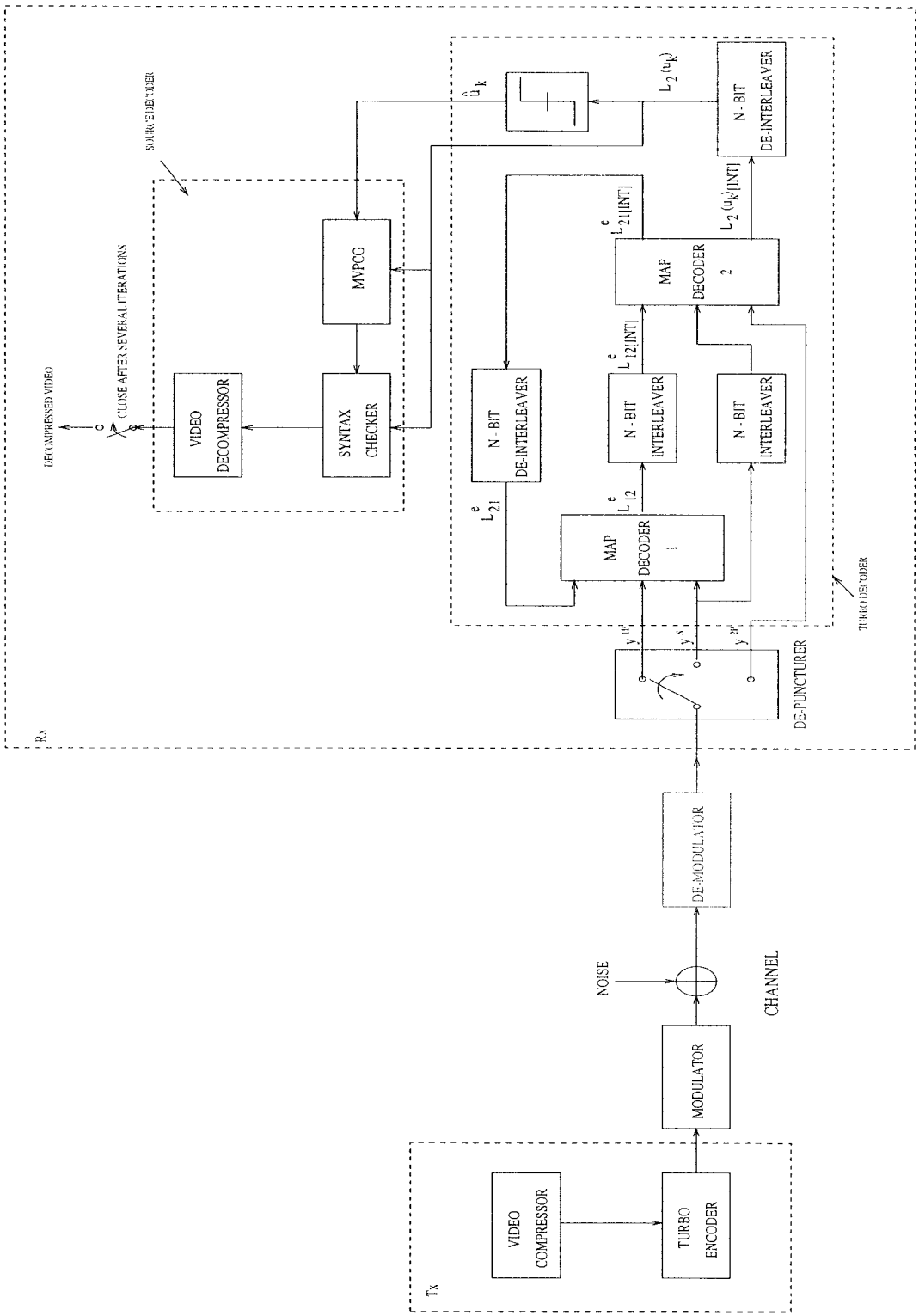

Figure 3.4: Block diagram of the SBECTC.

the hard decisions and soft outputs of the turbo decoder is investigated in Sub-section 3.3.3. Based on this relation, a Multiple Video Packet Candidates Generator (MVPCG) is developed, which is discussed in Sub-section 3.3.4. A Syntax Checker aiming to exploit the syntactic/semantic redundancy in an MPEG-4 compressed bitstream is introduced in Sub-section 3.3.5.

3.3.1 The Turbo Code Employed

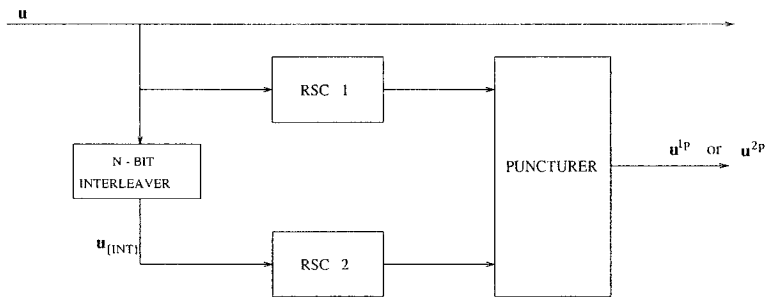

Figure 3.5: Turbo encoder ($\mathbf{u}^{1p}$ and $\mathbf{u}^{2p}$ represent the parity bit sequence from RSC encoder 1 and 2, respectively. $\mathbf{u}$ is information bit sequence.).

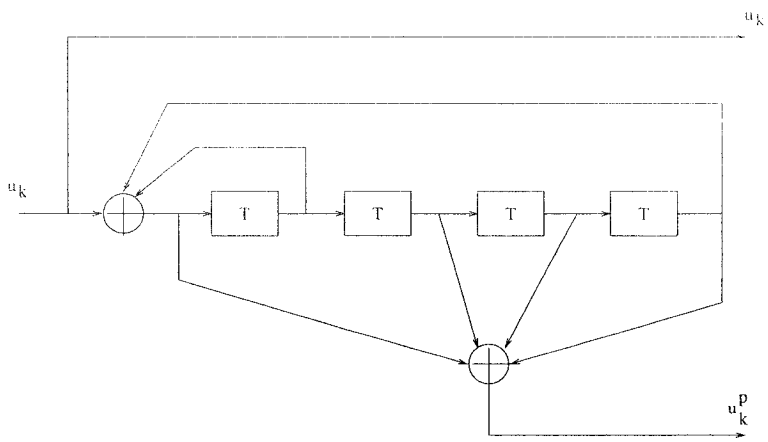

Figure 3.6: Recursive systematic convolutional (RSC) encoder (u_k is information bit and u_k^p is parity bit at time k).

The turbo code used in the current work is the one from [21]. Figure 3.5 shows the turbo encoder. The two RSC encoders have the same generator matrix [31 27] in

octal form, and the memory is 4. Figure 3.6 shows the RSC encoder. The puncture matrix is

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

so that the code rate of this turbo code is about $\frac{1}{2}$ (4 tail bits are added to restore the RSC encoder to all zero state for each turbo code block). A random interleaver with length N and the MAP decoding algorithm introduced in Section 2.2.4 are employed for this turbo code.

The bits on and below the VP from an MPEG-4 compressed bitstream, except the VP header bits, are encoded using this turbo code.

3.3.2 Channel and Modulator/Demodulator

As discussed in Section 1.1, there are three different kinds of transmission errors, i.e., random errors, burst errors and packet loss. In the current work, AWGN channel is considered resulting in random errors in the coded video bitstream. BPSK is employed for modulation [5].

3.3.3 Relation between the Hard Decisions and Soft Outputs from a Turbo Decoder

The relation between the hard decision on a bit and its corresponding soft output $L(u_k)$ from a turbo decoder is investigated, in order to take advantage of the available soft information. To find this relation, some simulations of turbo decoding are implemented. The block diagram of turbo decoding is as shown in Figure 2.5.

In the simulation, the turbo code employed is the one introduced in Sub-section 3.3.1. An AWGN channel and BPSK modulation are considered. The interleaver

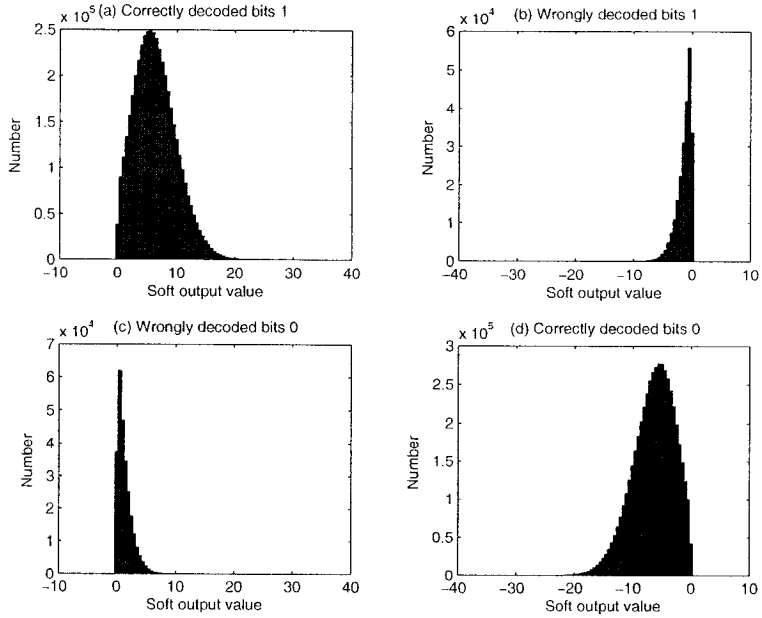

Figure 3.7: Histogram of soft outputs from turbo MAP decoder at iteration 1 and channel SNR 1.50 dB

length N is $1,004$ and channel SNR 1.5 dB. The information bit sequence is from an MPEG-4 compressed video. In the compression, a 6-second video "Table-Tennis" is used, which has frame size 352×240.

The histograms of the final outputs from the turbo decoder, which are the soft outputs $L_2(u_k)$ from the MAP decoder 2(please refer to Figure 2.5), are plotted for both correctly and wrongly decoded bits. The histograms show the soft outputs $L_2(u_k)$ of all the coded information bits.

Figure 3.7 shows the soft outputs at the first iteration. The soft outputs are classified into four categories as correctly decoded bits 1, correctly decoded bits 0, wrongly decoded bits 1 and wrongly decoded bits 0, respectively. Similarly, Figure 3.8 shows the histograms at the last iteration, i.e., the fifteenth iteration. It's noticed that the histograms from the third iteration are all similar to the ones at the fifteenth iteration and the only difference among them is that the number of correctly decoded bits is increased as the iteration goes on. For further investigation, Figure 3.9 presents all the soft outputs from all 15 iterations.

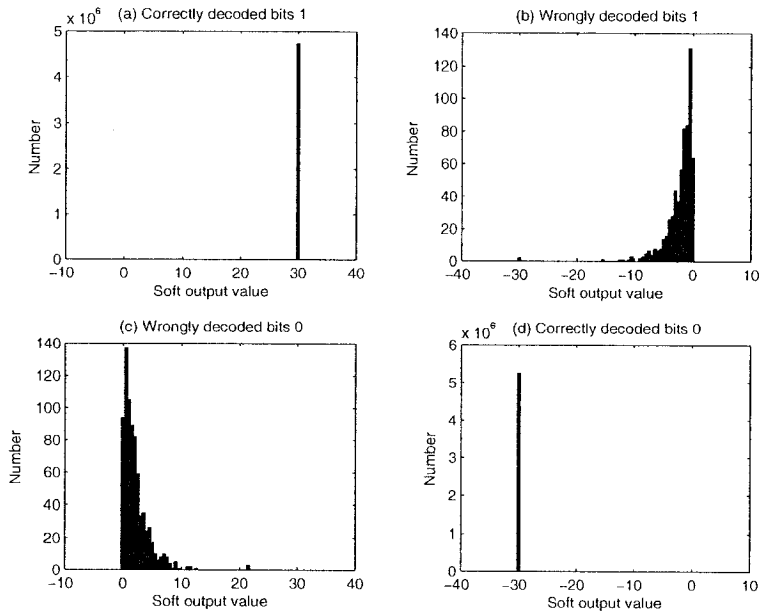

Figure 3.8: Histogram of soft outputs from turbo MAP decoder at iteration 15 and channel SNR 1.50 dB

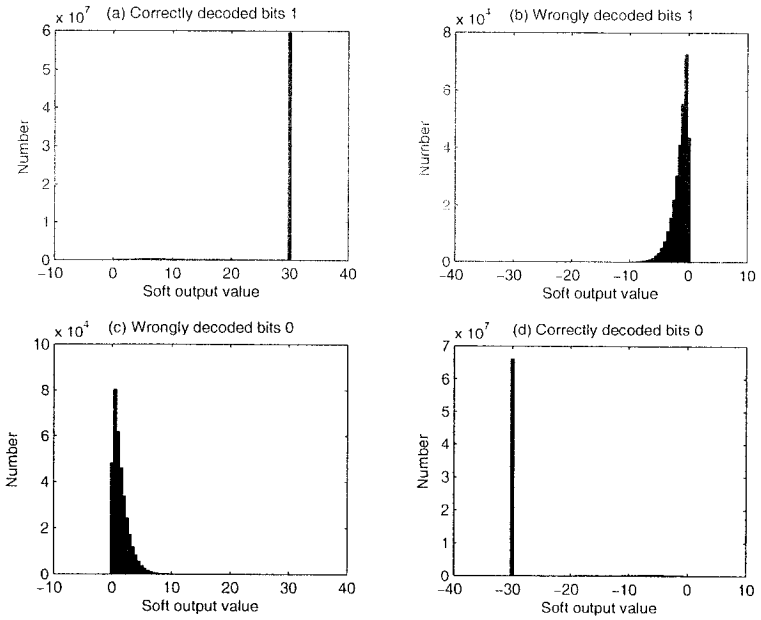

Figure 3.9: Histogram of soft outputs from turbo MAP decoder at all of iterations and channel SNR 1.50 dB

According to all these histogram figures, it can be observed that the soft outputs $L_2(u_k)$ of wrongly decoded bits generally are close to the threshold 0 (The definition of threshold is given by Equation 2.2.). Moreover, the average values of these four kinds of bits from all 15 iterations are calculated and the average values of correctly decoded bits 1, correctly decoded bits 0, wrongly decoded bits 1 and wrongly decoded bits 0 are 27.1210, -27.1208, -1.4977 and 1.5023, respectively.

From the simulation results presented above, it can be seen that the absolute soft output values $|L_2(u_k)|$ of wrongly decoded bits are much smaller than the correctly decoded bits. At this point, $|L_2(u_k)|$ can serve as reliability indicator of a bit's hard decision, i.e., the larger the $|L_2(u_k)|$ is, the more reliable is its hard decision, and vice versa.

3.3.4 Multiple Video Packet Candidate Generator

The Multiple Video Packet Candidate Generator (MVPCG) in Figure 3.4 is to make use of the available soft output $L_2(u_k)$ and hard decision of each bit from the turbo decoder to generate multiple VP candidates, based on the relation between the hard and soft outputs discussed in last sub-section. The MVPCG works on a VP basis. The number of multiple VP candidates generated by the MVPCG is fixed and denoted as N_{VP}. A method of constructing VP candidates, and an algorithm, Smallest Combination Algorithm (SCA) for finding N_{VP} VP candidates with the highest *a posteriori* probability (APP) as well as sorting them in order of decreasing APP, are developed.

The VP candidates are constructed as follows. When $|L_2(u_k)|$ of bit u_k is small, its APP is close to 0.5 according to Equation 2.2. Moreover, its corresponding hard decision is not reliable as discussed in Sub-section 3.3.3. Hence, it's reasonable to choose N_f bits with the smallest $|L_2(u_k)|$ value in a VP as Flip Bit Candidates (FBCs). By flipping some or all these N_f bits, VP candidates are generated. Considering all the combinations of these N_f bits, 2^{N_f} is the maximum number of available

VP candidates.

While constructing the VP candidates, the APP of each VP candidate is calculated based on the method described below.

1. Bit's APP

   Based on Equation 2.2, a bit's APP can be calculated as

$$P(u_k = 1|\mathbf{y}) = \frac{\exp[L(u_k)]}{1 + \exp[L(u_k)]} \tag{3.5}$$

$$P(u_k = 0|\mathbf{y}) = 1 - P(u_k = 1|\mathbf{y}) = \frac{1}{1 + \exp[L(u_k)]} \tag{3.6}$$

   Note that $L(u_k)$ is the soft outputs from MAP decoder 2, i.e., $L_2(u_k)$ and $\mathbf{y}$ is the received noisy information bit sequence.

2. Video packet's APP

   After receiving $\mathbf{y}$, the turbo decoder makes hard decisions on each transmitted bit u_k in a VP according to $L_2(u_k)$. The VP's APP is then the product of each bit's APP located in this VP under the assumption that the bits in the VP are independent to each other. This VP Candidate (VPC) is the MAP candidate from the view point of channel decoding (turbo decoding in the current work) since all the bits' APP are greater than 0.5. When flipping some or all the bits of the chosen N_f bits, the APP of the flipped bit is switched from $P(u_k = 1|\mathbf{y})$ to $P(u_k = 0|\mathbf{y})$ if the original hard decision is 1. Similarly, switch the APP from $P(u_k = 0|\mathbf{y})$ to $P(u_k = 1|\mathbf{y})$ if the original hard decision is 0. The bit APP can be calculated according to Equation 3.5 or Equation 3.6 and the original hard decision. Thus, the APP of a VP candidate can be calculated as

$$P(\text{VPC}_i|\mathbf{y}) = \prod_k P(u_k, \text{ or } \bar{u}_k|\mathbf{y}) \tag{3.7}$$

where i is the index of VP candidates, k is the bit index in a certain VP candidate and $\bar{u}_k$ represents the flipped version of bit u_k.

In Equation 3.7, the right hand side can be rewritten as the product of two parts, i.e.,

$$\begin{aligned} P(\mathrm{VPC}_i|\mathbf{y}) &= \prod_k P(u_k, \text{ or } \bar{u}_k|\mathbf{y}) \\ &= \prod_{k,k\neq L_j} P(u_k|\mathbf{y}) \cdot \prod_{k=L_j} P(u_k, \text{ or } \bar{u}_k|\mathbf{y}) \end{aligned} \tag{3.8}$$

where L_j represents the location of FBCs and j is the index of FBCs from 1 to N_f.

The algorithm, Smallest Combination Algorithm (SCA), is used to find N_{VP} VP candidates of the highest APP from the possible 2^{N_f} combinations and to sort these candidates in order of decreasing APP. This algorithm is important especially when N_f is a large number while N_{VP} is moderate.

To develop the SCA, Equation 3.8 is rewritten in logarithm scale as

$$\log P(\mathrm{VPC}_i|\mathbf{y}) = \sum_{k,k\neq L_j} \log P(u_k|\mathbf{y}) + \sum_{k=L_j} \log P(u_k, \text{ or } \bar{u}_k|\mathbf{y}). \tag{3.9}$$

The value of the first term in the right hand side of Equation 3.9 is fixed for all VP candidates since it's the summation of all the non-FBCs. Therefore, N_{VP} VP candidates that have the highest APP are found by identifying the N_{VP} combinations from N_f FBCs which may produce the largest summation of the second term in the right hand side of Equation 3.9.

After defining $P_{diff}(u_k)$ as

$$P_{diff}(u_k) \triangleq \log P(u_k|\mathbf{y}) - \log P(\bar{u}_k|\mathbf{y}) \tag{3.10}$$

the second term in the right hand side of Equation 3.9 becomes

$$\begin{aligned}
&\sum_{k=L_j} \log P(u_k, \text{ or } \bar{u}_k|\mathbf{y}) \\
&= \sum_{k=L_1, k\neq C_m}^{L_{N_f}} \log P(u_k|\mathbf{y}) + \sum_{k=C_m} \log P(\bar{u}_k|\mathbf{y}) \\
&= \sum_{k=L_1}^{L_{N_f}} \log P(u_k|\mathbf{y}) - \sum_{k=C_m} \log P(u_k|\mathbf{y}) + \sum_{k=C_m} \log P(\bar{u}_k|\mathbf{y}) \\
&= \sum_{k=L_1}^{L_{N_f}} \log P(u_k|\mathbf{y}) - \sum_{k=C_m} P_{diff}(u_k) \qquad (3.11)
\end{aligned}$$

where C_m represents the bit locations in a certain combination chosen from N_f $P_{diff}(u_k)$ and m is the index of the bits which are to be flipped. It should be noted that u_k is the original hard decision made by the turbo decoder and after flipping, u_k becomes $\bar{u}_k$. $P(u_k|\mathbf{y})$ and $P(\bar{u}_k|\mathbf{y})$ are available according to Equation 3.5 or 3.6.

In Equation 3.11, clearly, the first term is identical for all the VP candidates and the second term varies along with different flipped bits' combinations chosen from N_f FBCs. At this point, the identification of the N_{VP} combinations with the highest APP can be done by finding N_{VP} smallest combinations from N_f FBCs' logarithm probability difference $P_{diff}(u_k)$. Note that the VP candidates with the highest APP is the one directly from the turbo decoder without flipping any bit since it means the second term in the right hand side of Equation 3.11 is 0. Thus, only $N_{VP} - 1$ VP candidates need to be found by the SCA. The C program of the SCA is given in Appendix A.1. While finding the $N_{VP} - 1$ smallest combinations from N_f $P_{diff}(u_k)$, the SCA generates these candidates in such a way that the N_{VP} VP candidates are ordered from the smallest combination to the largest, corresponding to arrange these candidates from the highest APP to the lowest APP. Note that, before using the SCA, the N_f FBCs' logarithm probability difference $P_{diff}(u_k)$ are ordered increasingly. Finally, these N_{VP} VP candidates are sent to the next module, Syntax Checker.

3.3.5 Syntax Checker

The function of the Syntax Checker in Figure 3.4 is to check if a VP candidate has any syntactic/semantic error after receiving the multiple VP candidates generated by the MVPCG. The syntactic/semantic information exploited by the Syntax Checker is the syntax redundancy in an MPEG-4 compressed bitstream, as discussed in Section 3.1

The Syntax Checker works in the following way: Once a VP candidate without syntax violation is found, the Syntax Checker sends this candidate out and stops checking the rest of the VP candidates. Since the VP candidates are given in order of decreasing APP, the first VP candidate free from syntax violation is the best choice. If all the VP candidates can't pass the syntax checking, the one with the highest APP is accepted as the best candidate, i.e. the original VP from the turbo decoder without any flipping. In this way, the available source and channel information are combined together so that some improvement in terms of PSNR, number of error bits (NEB) or bit error rate (BER) can be expected.

It should be noted that if there are a large number of bit errors in a VP, the VP candidates provided by the MVPCG still have many bit errors. In this case, it is difficult for the Syntax Checker to find a VP candidate without syntax violation. Thus, the Syntax Checker only works effectively when there are a few errors in a VP.

3.4 Simulation Results

The performance of the SBECTC is tested against Turbo Decoding (TD). The difference between the TD and the SBECTC lies in that the former lacks the Syntax Checker and MVPCG shown in Figure 3.4 for the SBECTC. Thus, the output from turbo decoder is directly sent to the Video Decompressor for decompressing and no EC scheme is employed.

The current simulations work on a 6-second video sequence "Table-Tennis" which has frame size 352×240, frame rate 30 and bit rate 2 Mb/s. The simulations are run at channel SNR 3.0 and 3.5 dB, respectively. Only are intra-coded frames used in the compression. The total number of bits after compression is $9,873,776$. The interleaver length N of turbo code is $10,000$ and the iteration time is only 1. Table 3.1 shows the performance comparison, in terms of PSNR, BER and NEB, between two schemes for channel SNR 3.0 dB and Table 3.2 for channel SNR 3.5 dB. Also, the number of FBCs N_f varies to investigate the performance of the SBECTC. In the simulations, the selected VP candidates N_{VP} by MVPCG is 2^{N_f}. For comparison, Table 3.3 lists the possible maximum PSNR of Y,U and V component.

Table 3.1: BER, NEB and PSNR Vs. number of FBCs N_f at channel SNR 3 dB

	BER	NEB	PSNR (Y)	PSNR (U)	PSNR (V)
TD	5.22×10^{-4}	5150	9.363601	10.950125	8.956155
$N_f = 2$	4.27×10^{-4}	4213	15.425918	17.773954	16.289440
$N_f = 4$	3.88×10^{-4}	3831	17.743491	20.644899	19.280595
$N_f = 6$	3.64×10^{-4}	3598	19.395481	22.724291	21.404960

Table 3.2: BER, NEB and PSNR Vs. number of FBCs N_f at channel SNR 3.5 dB

	BER	NEB	PSNR (Y)	PSNR (U)	PSNR (V)
TD	5.30×10^{-5}	523	19.182158	22.173519	20.565707
$N_f = 2$	3.20×10^{-5}	316	28.637216	34.146479	32.891720
$N_f = 6$	2.83×10^{-5}	279	29.958338	36.081411	34.836587

According to the simulation results, the SBECTC has about 6.1 dB gain over the TD at channel SNR 3.0 dB in terms of PSNR (Y) even using only 2 FBCs and 4 VP candidates. The performance improvement is thus quite obvious. It's

Table 3.3: Maximum PSNR from the video compressor for "Table-Tennis"

PSNR (Y)	33.119841
PSNR (U)	37.340753
PSNR (V)	36.196230

also observed that the performance of the SBECTC, in terms of BER, NEB and PSNR, is improved along with the increased number of FBCs N_f. However, since the selected VP candidates $N_{VP} = 2^{N_f}$ and more VP candidates means more work of the Syntax Checker, the complexity is increased along with the increased N_f, too.

Table 3.4: BER, NEB and PSNR Vs. the order of choosing the best candidate at channel SNR 3.0 dB ($N_f = 6$, $N_{VP} = 64$)

	BER	NEB	PSNR (Y)	PSNR (U)	PSNR (V)
Choosing the highest APP	3.64×10^{-4}	3598	19.395481	22.724291	21.404960
Choosing the lowest APP	6.53×10^{-4}	6447	18.632484	22.355050	21.060311

Table 3.5: BER, NEB and PSNR Vs. the order of choosing the best candidate at channel SNR 3.5 dB ($N_f = 6$, $N_{VP} = 64$)

	BER	NEB	PSNR (Y)	PSNR (U)	PSNR (V)
Choosing the highest APP	2.83×10^{-5}	279	29.958338	36.081411	34.836587
Choosing the lowest APP	8.21×10^{-5}	811	29.124841	35.865247	34.454307

For comparison, simulations are also run under the same simulation condition as described at the beginning of this section, but choosing the best VP in a reverse

order, i.e., the candidate with the smallest VP APP is considered as the best candidate. Table 3.4 and Table 3.5 presents the performance comparison between these two different orders of choosing the best VP candidate at channel SNR 3.0 dB and 3.5 dB, respectively.

The simulation results in Table 3.4 and 3.5 show that the scheme of choosing the VP candidate with the highest APP works better than that of choosing the lowest APP in terms of PSNR, BER and NEB, as expected. Although choosing the best VP candidate with the smallest APP doesn't make sense, the simulation results can indeed show that wrongly decoded bits are most possibly the ones that have smaller $|L_2(u_k)|$ values. They are thus less reliable bits as stated in Section 3.3.4 since the VP candidate of the highest APP is constructed by flipping the bits with the smallest $|L_2(u_k)|$.

More simulation results of the SBECTC will be later presented in Chapter 4, in order to give a clearer comparison with the proposed scheme, iterative joint source and channel decoding using turbo codes, which is discussed in Chapter 4.

In the current chapter, some basic ideas of how to make use of soft outputs $|L_2(u_k)|$ of turbo decoder to assist error concealment in post-processing videos have been presented. First of all, $|L_2(u_k)|$ can be considered as a reliability measurement of the hard decision of the corresponding bit and thus the most uncertain bits in a VP can be picked up based on the magnitude of $|L_2(u_k)|$. Then, in the construction of VP candidates by flipping the most uncertain bits, $|L_2(u_k)|$ is used to calculate the APP of VP candidates. These ideas will be further used in the next chapter since they provide a reasonable way of using the information from the channel decoder. However, the SBECTC introduced in the current chapter is only a one-way method, i.e., information from the channel decoder is passed to the source decoder, but information extracted from the source is not fed back to the channel decoder. In the next chapter, the schemes of iterative joint source and channel decoding using turbo codes will be discussed, which will iteratively exchange information between

the channel decoder and the source decoder.

Chapter 4

Iterative Joint Source and Channel Decoding Using Turbo Codes

Chapter 3 has presented some ideas of how to use both the redundant information inherent in an MPEG-4 video bitstream and soft outputs from the turbo decoder to alleviate channel noise. It's well known that the most appealing feature of turbo codes is that they can achieve dramatic coding gain by *iteratively* exchanging extrinsic information between two constituent MAP decoders. The main purpose of this chapter is to make use of this iterative feature of turbo codes to *iteratively* send residual information in MPEG-4 compressed bitstreams to the turbo decoder to improve performance. Thus, this chapter investigates possible ways to transfer the residual information extracted by an MPEG-4 source decoder into soft values and feed back this information to the turbo decoder iteratively.

Chapter 4 is organized as follows: An overview of the proposed scheme, Iterative Joint Source and Channel Decoding (IJSCD), is given in Section 4.1. Feedback scheme and VP mixing/de-mixing, which are employed in the proposed scheme, are discussed in Section 4.2 and 4.3, respectively. The performance of the proposed scheme is assessed objectively and subjectively in Section 4.4. The complexity of the proposed scheme is also evaluated in Section 4.4. Section 4.5 presents a brief summary of this chapter.

4.1 Overview of Iterative Joint Source and Channel Decoding (IJSCD)

The whole system diagram of the proposed scheme, IJSCD, is shown in Figure 4.1, which consists of the modules: Video Compressor, VP Mixer (VPM), Turbo Encoder, Turbo Decoder, Modulator, De-Modulator, Multiple VP Candidates Generator (MVPCG), Syntax Checker, Modification Module (MM), VP De-mixer (VPDM) and Video Decompressor. Compared to the SBECTC, only the Modification Module, the VP Mixer and the VP De-mixer are new in the proposed scheme. The other modules are the same as discussed in Chapter 3.

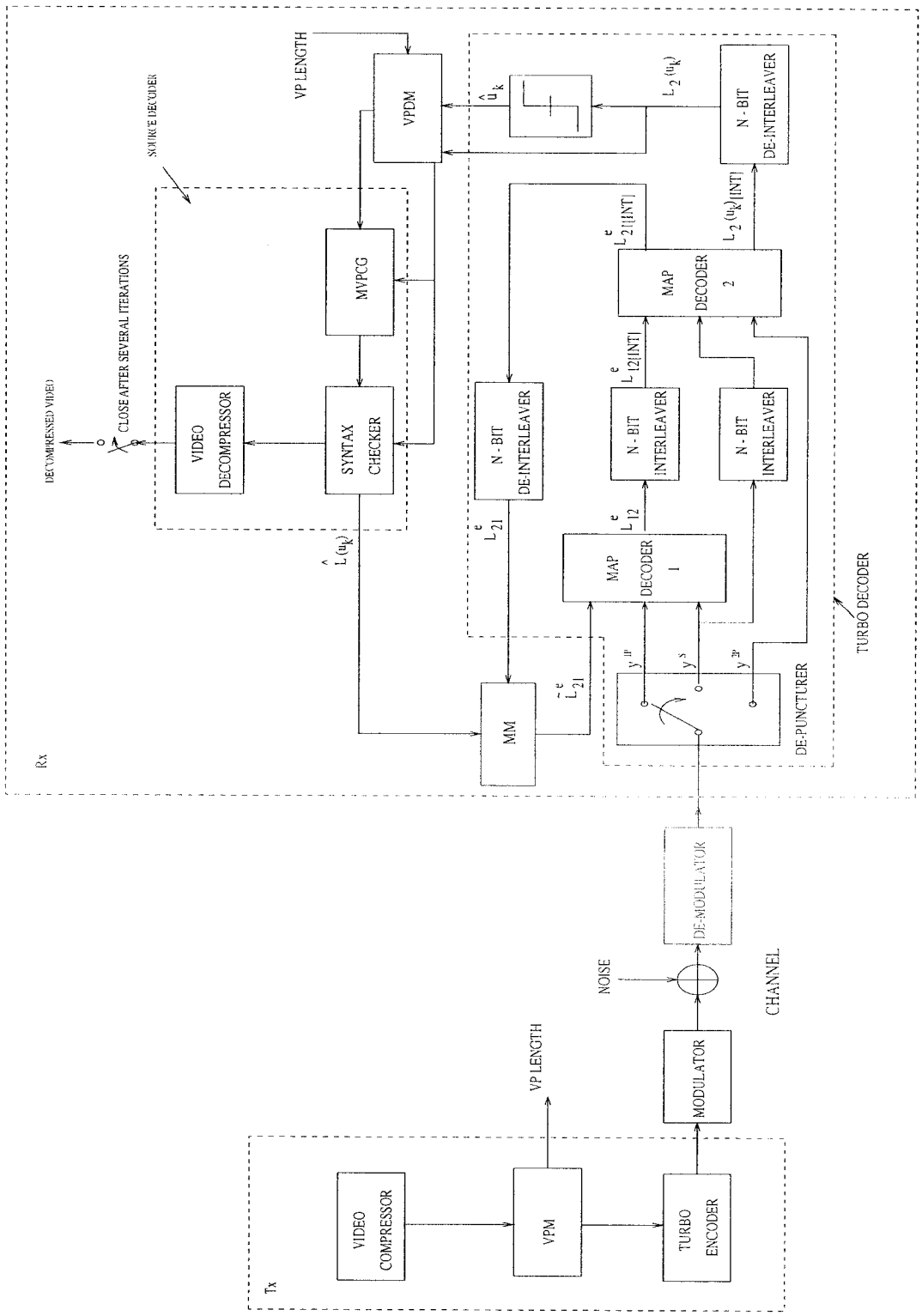

Figure 4.1: Block diagram of the proposed scheme, iterative joint source and channel decoding (IJSCD).

The proposed scheme, Iterative Joint Source and Channel Decoding (IJSCD), is developed on the basis of the SBECTC introduced in Chapter 3 while making use of the iterative feature of turbo decoding. The Syntax Checker and Multiple Video Packet Candidates Generator (MVPCG)(Please refer to Section 3.3.5 and 3.3.4) developed for the SBECTC are adopted by the proposed scheme and they have the same functionality as in the SBECTC. MVPCG takes in the soft output information from the turbo decoder and generates N_{VP} VP candidates for syntax checking in the Syntax Checker. Among N_{VP} VP candidates, the Syntax Checker chooses one as the best VP candidate if it has the highest VP probability and has no syntax violation.

In addition to the MVPCG and the Syntax Checker, the proposed scheme introduces a method of feeding back the best VP candidate information into the turbo decoder, which is done by the MM in Figure 4.1. The MM is further described in Section 4.2. Through three core parts of the proposed scheme, the MVPCG, the Syntax Checker and the MM, the interactivity between the channel decoder(turbo decoder) and the source decoder is established.

It's known that in some turbo coding blocks, there may still remain a large number of errors even after many iterations, which is considered as one of the convergence patterns of turbo decoding by some researchers [36, 37]. The convergence patterns of turbo decoding are thus described in Sub-section 4.3.1. As stated in Chapter 3, the Syntax Checker can only effectively handle a few bit errors in a VP given a moderate complexity. To deal with the turbo coding blocks with a large number of errors, the VP Mixer module at the transmitter side and the VP Demixer module at the receiver side are required. They're introduced in Sub-section 4.3.2.

Table 4.1: 8 combinations of transmitted bits, turbo decoded bits and source decoded bits

Original Bit (u_k)	Turbo Decoded		Source Decoded		Usual Effect of Feedback Scheme	Case No.
	Hard Decision $\hat{u}_k$	Sign of $L_2(u_k)$	Hard Decision	Sign of $\hat{L}(u_k)$		
1	1	+	1	+	Very Positive	1
			0	-	Negative	2
	0	-	1	+	Positive	3
			0	-	Very negative	4
0	1	+	1	+	Very negative	5
			0	-	Positive	6
	0	-	1	+	Negative	7
			0	-	Very Positive	8

4.2 Feedback Scheme

In this section, first, how the source information can affect turbo decoding is investigated. Then, based on the investigation, a feedback scheme is developed.

4.2.1 Usual Effect of Feedback scheme on Turbo Decoding

Before starting to design the scheme of feeding back the source information into the turbo decoder, how the source decoder can affect the turbo decoder must be examined.

The relationship among original/transmitted bits, turbo decoded bits and the signs of their corresponding soft values $L_2(u_k)$, as well as source decoded bits and the signs of their corresponding soft values $\hat{L}(u_k)$ are shown in Table 4.1. Based on the hard decisions from the Syntax Checker, the sign of the soft value $\hat{L}(u_k)$ in Table 4.1 is determined according to the same principle as applied to the turbo decoder, i.e., if a bit's hard decision is 1, its soft value is bigger than the threshold

0; otherwise, its soft value is less than 0.

There are 8 possible combinations with different effects on the turbo decoder as shown in Table 4.1. As described in Chapter 3, the output bits from the Syntax Checker are mostly the same as the ones from the turbo decoder and the different bits are only the flipped ones. Thus, Cases 1 and 8 in Table 4.1 occur with the highest probability. Also, since the Syntax Checker can further correct the remaining error bits after turbo decoding, Cases 3 and 6 is helpful for the turbo code error performance. Cases 2 and 7 occur when bits are wrongly flipped. However, these two cases won't cause any syntax error and thus the video quality won't degrade too much. Cases 4 and 5 happen in case that the turbo decoder makes wrong decisions and the Syntax Checker fails to catch these errors. There are two reasons why the Syntax Checker fails to catch these errors. One is that too many errors are located in a VP, which is beyond the capability of the Syntax Checker. The other is that these errors don't result in syntax errors.

In summary, if the feedback scheme can generate the source decoded soft decisions ($\hat{L}(u_k)$) as presented in Table 4.1 , it will be helpful in terms of video quality and may be helpful in terms of BER, which need to be proved by simulations.

4.2.2 Modification Module

According to soft value $\hat{L}(u_k)$ discussed in last sub-section, it's a natural idea that the feedback scheme can be a kind of modification which works only on $L_{21}^e(u_k)$ (see Figure 4.1) since the information exchanged between the two MAP decoders is only extrinsic information $L_{ij}^e(u_k)$ ($i,\ j = 1,\ 2$, representing MAP decoder 1 or 2) and the output hard decision on each bit is from MAP decoder 2. The modification combines together $L_{21}^e(u_k)$ and $\hat{L}(u_k)$ as

$$\tilde{L}_{21}^e(u_k) = L_{21}^e(u_k) + \hat{L}(u_k), \tag{4.1}$$

which is fed back to the MAP decoder 1.

The modification works only on the best VP candidate which is considered as error-free by the Syntax Checker. Note that the method of generating VP candidates and choosing the best VP candidate is the same as described in Section 3.3.4 and 3.3.5. The soft value $\hat{L}(u_k)$ (called Modification Value hereinafter) can be chosen from M1, M2 and M3 according to the hard decisions made by the Syntax Checker and the relations described in Table 4.1 as follows:

- Among the FBCs, if the decision on bit u_k made by the Syntax Checker agrees with $\hat{u}_k$ made by the turbo decoder, $M1$ will be chosen. The Modification Module (MM) works as

$$\hat{L}(u_k) = \begin{cases} M1 & \text{if } \hat{u}_k = 1 \\ -M1 & \text{if } \hat{u}_k = 0 \end{cases} \tag{4.2}$$

  which covers Cases 1, 8, 4 and 5 in Table 4.1.

- Among the FBCs, if the decision on bit u_k made by the Syntax Checker doesn't agree with $\hat{u}_k$, $M2$ will be chosen. The MM works as

$$\hat{L}(u_k) = \begin{cases} -M2 & \text{if } \hat{u}_k = 1 \\ M2 & \text{if } \hat{u}_k = 0 \end{cases} \tag{4.3}$$

  which covers Cases 3, 6, 2 and 7 in Table 4.1.

- If u_k doesn't belong to the FBCs, $M3$ will be chosen and the MM works as

$$\hat{L}(u_k) = \begin{cases} M3 & \text{if } \hat{u}_k = 1 \\ -M3 & \text{if } \hat{u}_k = 0 \end{cases} \tag{4.4}$$

  which covers Cases 1, 8, 4 and 5 in Table 4.1.

All Mi's (i = 1, 2, 3) are positive numbers and will be decided empirically. If all VP candidates don't pass syntax checking, no modification will be done for this VP. In this case, the hard decision on each bit is the same as the one from the tubo decoder and the soft value of each bit keeps intact.

The modification work described above is similar to the weight operation in [22]. However, the work in [22] on MPEG-1 videos only considers syntax information from robustly recovered start codes. Other than that, no more syntax information are used.

4.3 VP Mixing/De-mixing

In this section, the convergency patterns of MAP turbo decoding are discussed. A VP mixing/de-mixing scheme is introduced to deal with one of the convergency patterns, turbo coding blocks with a large number of errors.

4.3.1 Convergence Patterns of MAP Turbo Decoding

In the simulations of MAP turbo decoding, it is noticed that there are some turbo coding blocks with a large number of errors even after many iterations. The tubo coding blocks with a large number of errors are also observed by other researchers as one of the convergence patterns of MAP turbo decoding [36, 37]. In [36], Takeshita *et al.* address the following convergence patterns of turbo decoding:

1. *A frame converges either to zero errors or to a small number of errors (residual errors), possibly exhibiting a small jitter in the number of errors, after a moderate number of iterations. (This occurs in the large majority of cases.)*

2. *A frame maintains a large number of errors for many iterations (possibly many times larger than the average) before converging to zero errors.*

3. *A frame has an oscillating number of errors, but with a minimum number of errors that is very small (on the order of the number of residual errors). The oscillations may stop after a while or continue indefinitely.*

4. *A frame converges to a large number of errors (about 10% of the information bits).*

5. *A frame maintains a large number of errors (about 10% of the information bits) with some jitter indefinitely without converging.*

Note that, in the above description,"frame" has the same meaning as "turbo coding block" in this thesis and "frame" in this thesis refers to a time sample of video sequences. Both Cases 4 and 5 in the above list induce the turbo coding blocks with a large number of errors. Cases 2 and 3 may induce the turbo coding blocks with a large number of errors depending on how many iterations a turbo decoding process runs.

The convergence patterns of turbo decoding don't obviously affect the BER of a turbo code since the majority of turbo coding blocks converge either to zero errors or to a small number of errors. When the turbo code is employed to protect a bit sequence of a compressed video, the turbo coding blocks with a large number of errors can have a dramatic effect on the video quality.

4.3.2 VP Mixer and VP De-mixer

Last sub-section described the convergence patterns of MAP turbo decoding. In the convergence patterns, the turbo coding blocks with a large number of erroneous bits give rise to a challenge for successful design of iterative joint source and channel decoding scheme using turbo codes. The VP Mixer (VPM) and De-Mixer (VPDM) in the proposed scheme are aimed at addressing this problem.

The functionality of the VPM in the proposed scheme is to distribute the bits from a VP into several different turbo coding blocks before turbo encoding. Accordingly, the bits in a turbo coding block must be recovered into their original VPs at the receiver side, which is done by the VPDM in Figure 4.1. In this way, a large number of errors in a turbo coding block are located in different VPs so that the Syntax Checker can detect these errors. In VP mixing, the number of bits inserted into a turbo coding block from the same VP is limited by parameter N_m, i.e., the maximum number of bits from the same VP is N_m. Note that, VP mixing

introduced here is a common way to re-distribute error bits in order to easily catch these errors, as used in the previous work [27].

Some simulations are implemented to measure the actual performance of VP mixing at channel SNR 1.5 and 1.6 dB. In the simulations, the 6-second video "Table-Tennis" is used, which has a frame size 352×240 and is interlaced with 30 frames/second. It is compressed using MPEG-4 at bit rate 2 Mb/s and each row in a frame has 4 VPs. The turbo code used is the same as the one in Section 3.3.1, but with interleaver length 1004. The number of FBCs, N_f is chosen as 4 and the number of VP candidates, N_{VP} is 16. The modification parameters $M1$, $M2$ and $M3$ are chosen as 3.0, 0.5 and 0.5, respectively. The number of bits from the same VP in a turbo coding block is chosen as 20.

As an illustrated example,the turbo coding blocks with a large number of errors when the scheme, Turbo Decoding(TD), is used (for fair comparison, VP mixing/de-mixing is also employed by the TD) are listed in Tables 4.2 and 4.3. Tables 4.2 and 4.3 also list the number of error bits for the same turbo coding blocks when the proposed scheme is employed. Through these two tables, it's clear that VP mixing can indeed correct most of the turbo coding blocks with a large number of error bits.

The trade-off for the decreased number of tubo coding blocks of a large number of errors is the increased complexity, overhead bits and frame delay. The increased complexity by VP mixing/de-mixing is negligible compared with the interleaving/de-interleaving operation and MAP decoding algorithm of turbo codes. The overhead bits are used to send the length of each VP in order to reassemble each VP from the received turbo coding blocks at the receiver side. The VP length is sent together with other header bits of a VP using FLC coding. The frame delay is induced since the source decoder can only work after receiving all the bits in a frame. The overhead bits and frame delay will be discussed in detail in the next section.

Table 4.2: Turbo coding blocks with a large number of errors at channel SNR 1.5 dB after 15 iterations

Turbo coding block No.	TD	IJSCD
1246	62	0
1375	83	105
2163	114	127
4122	39	0
4651	51	64
5342	115	101
5850	91	0
6730	86	0
7569	15	0
7729	17	0
9857	53	0

Table 4.3: Turbo coding blocks with a large number of errors at channel SNR 1.6 dB after 15 iterations

Turbo coding block No.	TD	IJSCD
3790	124	0
9330	106	0
9897	124	0

4.4 Performance Evaluation

In this section, the performance of the proposed scheme, IJSCD is evaluated against the other two schemes, Tubo Decoding (TD) and Syntax Based Error Concealment using Turbo Codes (SBECTC). The TD is a simple scheme, in which the decoded bitstream from the turbo decoder is directly sent to the Video Decompressor and there is no error concealment. The SBECTC is introduced in Chapter 3, which, at the last iteration, makes use of the soft information provided by the turbo decoder for syntax based error concealment in the source decoder. It should be pointed out that the VP mixer/de-mixer is also employed in the TD and the SBECTC to keep the input bitstream to the turbo encoder identical for all three schemes.

Before evaluating the performance of the proposed scheme, the parameters associated with the proposed scheme, IJSCD, have to be determined at first. The parameter determination is discussed in Section 4.4.1.

The performance of the proposed scheme, IJSCD, is assessed objectively and subjectively. The objective measure includes PSNR, BER and NEB. The simulation results of objective performance evaluation is presented in Section 4.4.2. Since rigorous subjective measurement is costly and time-consuming as discussed in Section 1.3, this thesis provides some frames processed by different schemes for the readers' own evaluation. Also, comments on subjective quality as observed by the author are given. The results of subjective performance evaluation are presented in Section 4.4.3.

In addition to evaluating the objective and subjective performance, the complexity of the proposed scheme is also assessed against the other two schemes. The evaluation results of complexity are described in Section 4.4.4

All the performance evaluation results are obtained through simulations. The simulations work on two 6-second videos,"Table-Tennis" and "Football", which have the frame size of 352 $\times$ 240 pixels and are interlaced with 30 frames/second, respectively. The videos are compressed using MPEG-4 at bit rate 2 Mb/s. In the

MPEG-4 compression, each row has 4 VPs and thus the video has 10,860 VPs. The Group of Video Object Plane (GOV) layer is employed in the compression and it has 15 frames with 1 I-frame, 4 P-frames and 10 B-frames as shown in Figure 2.2. After compression, the total number of bits in the compressed MPEG-4 bitstream of "Table-Tennis" is 10,528,416. Hence, the compression rate is about 17.43. For "Football", the total number of bits is 10,312,136 and the compression rate is about 17.79.

The turbo code employed is the same as that presented in Section 3.3.1 but with interleaver length 1004. Thus, the number of bits in a turbo code block is 1004, too. In turbo encoding, only the bits in or below the VP layer and after the VP header bits are encoded. Other un-coded bits are assumed to be received correctly. In this case, for "Table-Tennis", the total number of turbo coding blocks is 10,084 and turbo encoded bits are 10,083,513. The total number of turbo coding blocks is 9,868 and turbo encoded bits are 9,867,198 for "Football".

In the simulations, for "Table-Tennis", the channel SNR ranges from 1.3 dB to 1.8 and the simulations are actually done at channel SNR 1.3, 1.4, 1.5, 1.6, 1.7 and 1.8 dB, respectively. For "Football", the simulations are actually run at channel SNR 1.5 and 1.6 dB. Each set of simulation parameters is run 5 times and the results shown in this section are the average value of all these 5 experiments after 15 iterations. The PSNR value is the average of all 181 frames.

All the simulations are implemented using C/C++ based on the MPEG-4 reference software, MoMuSys, and run on the Sun system (UNIX). The decompressed video sequences are also played on the Sun system to watch their performance subjectively.

4.4.1 Parameter Determination

The parameters associated with the proposed scheme, IJSCD, includes:

- Number of FBCs, N_f

- Number of VP candidates, N_{VP}. N_f and N_{VP} are introduced in Section 3.3.4.
- Modification parameters $M1$, $M2$ and $M3$ introduced in Section 4.2.
- Number of bits from the same VP in a turbo coding block, N_m, described in Section 4.3.2.

N_{VP} has a significant effect on both the complexity and performance of the proposed scheme since it directly decides how many VP candidates need to be tried, and the bigger the N_{VP} is, the greater the probability is that the selected VP candidate is of high quality with fewer bit errors. In the simulations, N_{VP} is chosen as 16 and 64. Corresponding to that N_f is chosen as 4 and 6, respectively. N_f, $M1$, $M2$, $M3$ and N_m have no significant effect on the complexity of the proposed scheme. However, their effect on the performance of the proposed scheme needs to be investigated. N_m can affect frame delay and overhead bits as discussed in Section 4.3.2. Thus, the choice of N_m is based on the acceptable delay and overhead bits as well as its effect on the performance of the proposed scheme. N_f, $M1$, $M2$ and $M3$ are determined according to the experiment results of evaluating their effect on the performance of the proposed scheme.

Note that only video "Table-Tennis" is used in the experiments for determination the parameters associated with the proposed scheme. Further empirical investigation may allow the parameter values determined in this section to be refined.

1. Determination of the number of FBCs (N_f)

As stated in Section 3.3.4, the maximum N_{VP} is 2^{N_f} after N_f is set up. The purpose of determination of N_f is to investigate what the effect is on the performance of the proposed scheme when making $N_{VP} > N_f \geq log_2 N_{VP}$. To do this, the number of VP candidates N_{VP} is fixed as 16. When investigating the effect of the number of FBCs N_f on the performance of the proposed scheme, IJSCD, N_f is chosen as 4, 10 and 15, respectively.

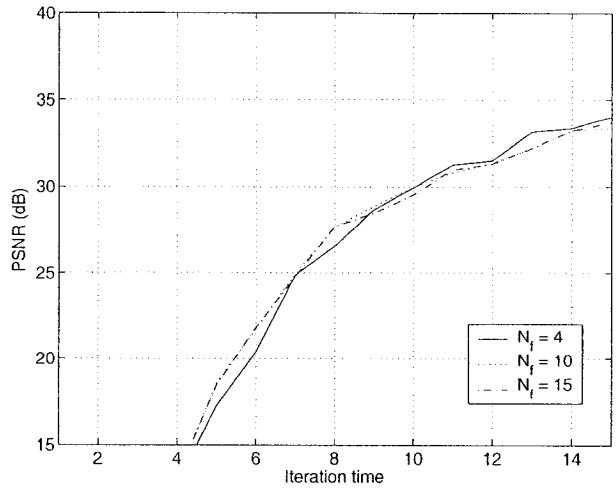

Figure 4.2: Luminance PSNR (Y) Vs. number of FBCs N_f at channel SNR 1.50 dB. N_{VP} is fixed as 16.

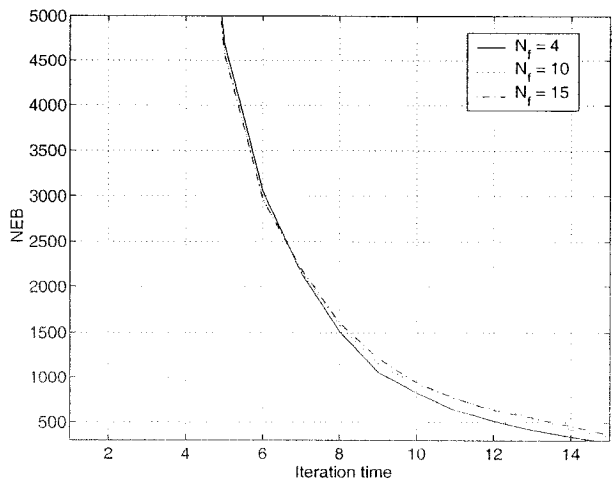

Figure 4.3: Number of error bits (NEB) Vs. number of FBCs N_f at channel SNR 1.50 dB. N_{VP} is fixed as 16.

In the experiments, $M1$, $M2$, $M3$ and N_m are fixed as 3.0, 0.5, 0.5 and 20, respectively. The experiments are implemented at channel SNR 1.50 dB and the other simulation conditions are the same as those described at the beginning of Section 4.4. Each set of experiments is also run 5 times. The experiment results, in terms of luminance PSNR (Y) and NEB, are shown in Figure 4.2 and 4.3, respectively. The experiment results, in terms of chrominance PSNRs, are similar to that of luminance PSNR, so they are not presented.

In Figure 4.2, it's observed that there are no obvious performance improvement by increasing N_f from 4 to 10, and further to 15, in terms of PSNR, while keeping N_{VP} fixed as 16. In Figure 4.3, the performance with $N_f = 4$ is better than other two cases in terms of NEB. Thus, in later simulations, N_{VP} is set at first according to acceptable complexity, and N_f is then chosen as $log_2 N_{VP}$. The simulation results indicates that the most uncertain bits in a VP are indeed related to the bits with the smallest $|L_2(u_k)|$ as hypothesized in Section 3.3.4.

2. Determination of $M1$, $M2$ and $M3$

The purpose of determination of $M1$, $M2$ and $M3$ is to choose a combination of M_i's$(i = 1, 2, 3)$ so that the proposed scheme can achieve the best performance. The experiments are carried out at channel SNR 1.50 dB. For each combination of M_i's, the experiment runs only once. In the experiments, the other parameters N_f, N_{VP} and N_m are fixed as 4, 16 and 20, respectively. The other simulation conditions are the same as those listed at the beginning of Section 4.4.

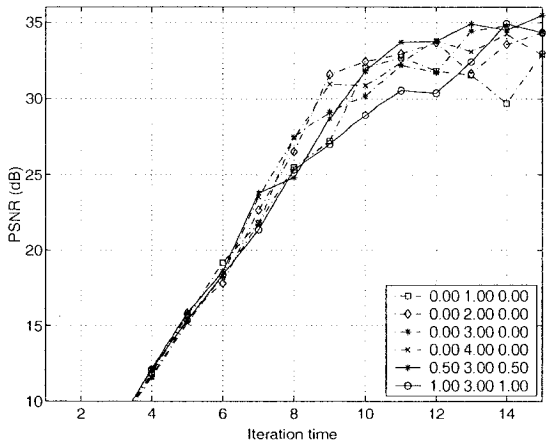

Figure 4.4: Luminance PSNR (Y) Vs. modification parameters $M1$, $M2$ and $M3$ at channel SNR 1.50 dB(The numbers in the figure is in the order of $M2$, $M1$ and $M3$).

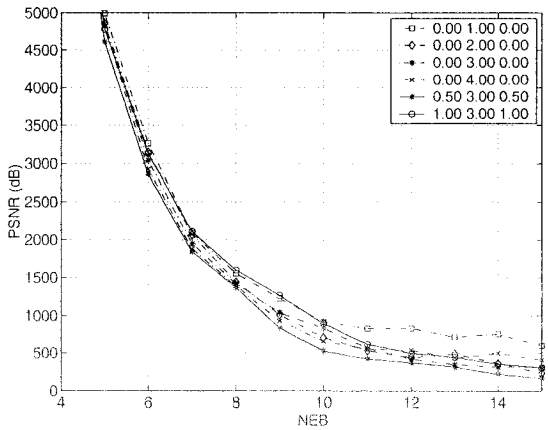

Figure 4.5: Number of error bits (NEB) Vs. modification parameters $M1$, $M2$ and $M3$ at channel SNR 1.50 dB(The numbers in the figure is in the order of $M2$, $M1$ and $M3$).

The performance of the proposed scheme, IJSCD, with various combinations of $M1$, $M2$ and $M3$, in terms of luminance PSNR (Y) and NEB, are shown respectively in Figure 4.4 and 4.5. The values of $M1$, $M2$ and $M3$ corresponding to each curve are also listed in these two figures. Since the experiment results, in terms of chrominance PSNR (U) and (V), are similar to that of luminance PSNR, they are not shown for brevity. From these two figures, it's observed that the performance of the proposed scheme with the combination of $M1 = 3.00$, $M2 = 0.50$ and $M3 = 0.50$, in terms of PSNR, is the best among all the combinations of M_i's, beginning from iteration 11. In terms of NEB, the proposed scheme with the combination of $M1 = 3.00$, $M2 = 0.50$ and $M3 = 0.50$ achieves the best performance for all 15 iterations. Hence, $M1$, $M2$ and $M3$ in the proposed scheme will be respectively set as 3.00, 0.50, and 0.50. Note that further empirical investigation may refine $M1$, $M2$ and $M3$ chosen here.

3. Determination of N_m

The number of bits from the same VP in a turbo coding block(N_m) used in the VPM (Please refer to Figure 4.1) is determined according to the acceptable frame delay, overhead bits and its effect on the performance of the proposed scheme. In the experiments, N_m is chosen as 20, 50 and 100, respectively. The other parameters N_f, N_{VP}, M_1, M_2 and M_3 are fixed as 4, 16, 3.0, 0.5 and 0.5, respectively. The experiments are implemented at channel SNR 1.6 dB and the other simulation conditions are the same as presented at the beginning of Section 4.4.

First, the frame delay caused by VP mixing is investigated. When turbo coding/decoding is employed and VP mixing is not used, the compressed bitstream is written directly into each turbo coding block in the order they are generated. Once a turbo coding block is filled, it can be sent over the channel. In this case, the frame delay caused by blocking bits is trivial since the block length (1,004-bit) is short, when considering that the bit rate is 2 Mb/s and a VP has several

Table 4.4: Maximum frame delay caused by VP mixing Vs. number of bits from the same VP (N_m) in a turbo coding block

N_m	Maximum Frame Delay
20	5
50	1
100	0

hundreds/thousands of bits. However, if the block length is too large, the attention has to be paid to the delay resulted from blocking bits. When the proposed scheme is employed, one VP may be encoded into many turbo coding blocks. The delay caused by VP mixing thus has to be measured carefully.

In particular, during the turbo encoding process, the index of the last turbo coding block for transmitting a VP is written down for each VP. This index is denoted as V_{ij} where i is the frame index and j is the VP index, When measuring the frame delay, the maximum V_{ij} in a frame (denoted as F_i, where i is the frame index) is found and saved for each frame. It's known that a VP cannot be decoded until all the bits in a VP are received and a frame cannot be decompressed until all the bits in this frame are received. Frame delay (denoted by D_i, where i is the frame index) is generated if all the bits in the current frame are received while not all the bits in its past frames have been received. The frame delay D_i is measured as the frame distance between the current frame i and its past frame i_p, which is the farthest frame resulting in $F_{i_p} \geq F_i$. The maximum D_i is the maximum frame delay. The experiment results are shown in Table 4.4.

Table 4.4 shows that the maximum frame delay decreases as the number of bits from the same VP N_m in a turbo coding block increases. The maximum frame delay is 5 and thus $\frac{1}{6}$ second, which is acceptable for non-interactive applications. Note that B-frames (which are used in the simulations), are also not generally used in interactive applications.

In addition to frame delay, the number of overhead bits caused by VP mixing also needs to be calculated. As mentioned in Section 4.3.2, each VP's length is sent as side information together with the VP's header bits using FLC coding so that the mixed bits can be recovered into their original VPs. Like other bits such as the bits in a syntax layer higher than the VP, the overhead bits for sending VP length are also assumed to be received correctly.

Given the current simulation conditions described at the beginning of Section 4.4, the total bits after compression are $10,528,416$ and the total number of VPs is $10,860$. It's noticed that, under the current compression conditions, each VP contains several thousands of bits, which requires 13 or 14 bits for coding the length of a VP. Thus, the overhead bits account for about 1.34 - 1.44% of the compressed bitstream. It should be noted that the number of overhead bits are not affected by N_m.

After considering the frame delay caused by N_m and overhead bits required by VP mixing, the effect of N_m on the performance of the proposed scheme, in terms of PSNR, BER and NEB, is evaluated when N_m is chosen as 100, 50 and 20, respectively. Since the output bitstream from the Video Compressor in Figure 4.1 is fixed while N_m changes, the bitstreams sent over the channel are different for three different N_m's. Hence, to evaluate the effect of N_m, it's necessary to compare the performance of the proposed scheme with the other two decoding schemes, the TD and the SBECTC, to see which N_m can result in the best performance improvement.

Table 4.5: Performance versus number of bits from the same VP (N_m) after 15 iterations at channel SNR 1.6 dB

N_m	Scheme	BER	NEB	PSNR (Y)	PSNR (U)	PSNR (V)
100	TD	3.76×10^{-5}	396.2	34.238	39.204	38.311
	SBECTC	3.73×10^{-5}	392.6	34.376	39.277	38.436
	IJSCD	4.02×10^{-5}	423.6	34.384	39.264	38.438
50	TD	2.87×10^{-5}	302.2	32.218	36.948	35.924
	SBECTC	2.76×10^{-5}	292.2	32.614	37.223	36.254
	IJSCD	2.15×10^{-5}	226.2	32.996	37.546	36.717
20	TD	3.78×10^{-5}	398.4	32.378	37.048	36.05
	SBECTC	3.43×10^{-5}	361.2	33.744	38.587	37.679
	IJSCD	9.35×10^{-6}	98.4	35.501	40.052	39.33

Each set of experiment is run 5 times. Table 4.5 shows the average results of all 5 experiments. When N_m is 20, the proposed scheme raises PSNR (Y) by 3.123 dB and the number of bit errors is decreased from 398.4 to 98.4 compared to the TD. Table 4.5 shows the performance improvement while N_m decreases as expected. Since, when N_m is chosen as 20, the frame delay is acceptable and the best performance can be achieved comparing to that $N_m = 50$ and $N_m = 100$, N_m is set as 20 in the proposed scheme.

In summary, the parameters associated with the proposed scheme, IJSCD, are determined as follows: The modification parameters $M1$, $M2$ and $M3$ are chosen as 3.00, 0.50 and 0.50, respectively. The number of VP candidates N_{VP} is set according to the acceptable complexity and can be chosen as 16 or 64. The number of FBCs N_f is chosen as $log_2 N_{VP}$ after N_{VP} has been set up. The number of bits from the same VP in a turbo coding block N_m in VP mixing is set as 20.

4.4.2 Objective Performance

To evaluate the performance of the proposed scheme, PSNR, BER and NEB are chosen as the objective measures. The performance evaluation is carried out by simulations. In the simulations, two videos, "Table-Tennis" and "Football", are used. The number of VP candidates (N_{VP}) is chosen as 16 and 64 so that the corresponding N_f is 4 and 6, respectively. The other parameters required by the proposed scheme are set up as discussed in Section 4.4.1. The other simulation conditions are the same as described at the beginning of Section 4.4.

For performance comparison, the SBECTC and the TD are also simulated at the same simulation conditions. It should be noted again that the VPM/VPDM is also applied to these two schemes to keep the turbo encoded bitstream identical for all three schemes.

1. Objective performance when Video "Table-Tennis" employed

The simulations are actually run at channel SNR 1.3, 1.4, 1.5, 1.6, 1.7 and 1.8 dB. Each set of simulation parameters are run 5 times and the simulation results shown in this part are the average values of these 5 experiments.

Figures 4.6, 4.7 and 4.8 shows the objective performance comparison, in terms of luminance PSNR (Y), blue chrominance PSNR (U) and red chrominance PSNR (V), among three schemes, the TD, the SBECTC and the proposed scheme, IJSCD. The performance comparison, in terms of bit error rate (BER) and number of error bits (NEB) in the recovered bit sequences, are shown in Figures 4.10 and 4.9. For comparison, bit error rate P_b in case that no channel coding is applied for the compressed bitstream is calculated according to Equation 4.5 [5] (note, this equation can be applied to BPSK and QPSK modulations.) and presented in Figure 4.10.

$$P_b = Q\left(\sqrt{\frac{2E_b}{N_0}}\right) \tag{4.5}$$

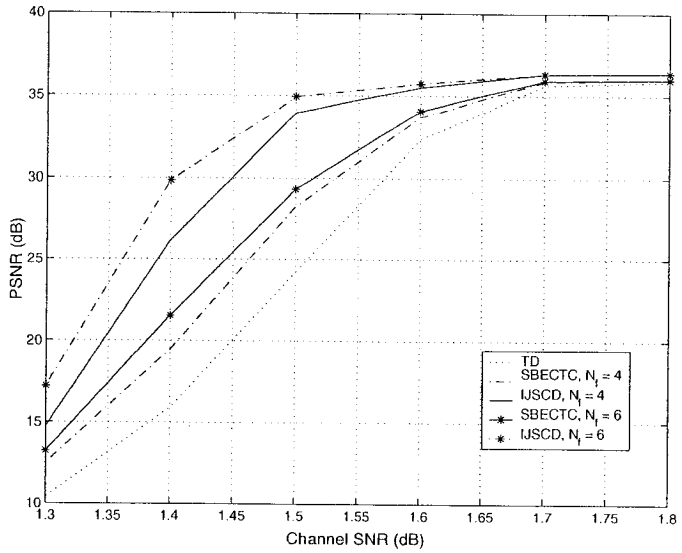

Figure 4.6: Luminance PSNR (Y) Vs. channel SNR and the number of FBCs N_f

Figures 4.6, 4.7 and 4.8 clearly show that the proposed scheme indeed has significant performance improvement in terms of PSNR and the performance is further improved as the number of FBCs N_f is increased. At channel SNR 1.4 dB, the proposed scheme with $N_f = 6$ can achieve about 30 dB in PSNR (Y) and at this PSNR, the reconstructed video is viewable. Unlike that, PSNR (Y) of the reconstructed videos by the SBECTC and the TD is below 25 dB, which is not acceptable. At channel SNR 1.5 dB, PSNR (Y) of the reconstructed videos by the TD is still below 25 dB and PSNR (Y) achieved by the SBECTC with $N_f = 6$ is about 29 dB. In contrast, the proposed scheme with $N_f = 6$ can get about 35 dB of PSNR (Y). At channel SNR 1.6 dB and N_f 4, the proposed scheme can achieve 35.501 dB in terms of luminance PSNR (Y), which is 3.123 dB higher than the TD and 1.757 dB higher than the SBECTC. Similar performance is also achieved in terms of red chrominance and blue chrominance as presented in Figures 4.7 and 4.8. At channel SNR 1.7 and 1.8 dB, the performance achieved by three schemes, in terms of PSNR, are quite close because the actual BER is very low, which is in the order of 10^{-6} as shown in Figure 4.10.

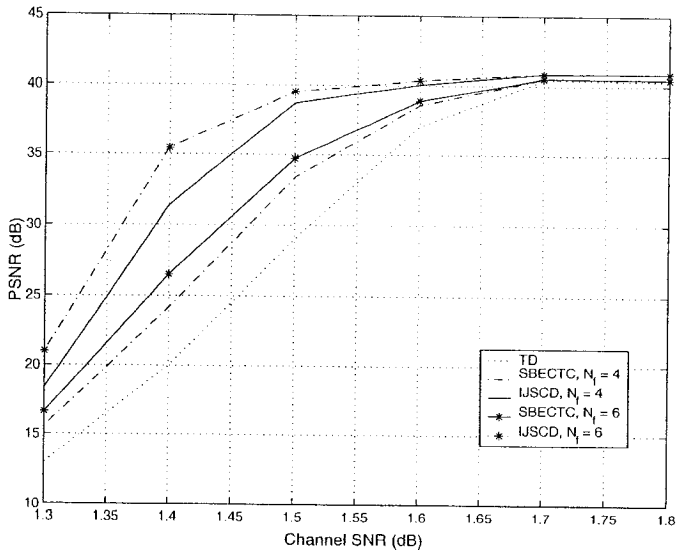

Figure 4.7: Blue chrominance PSNR (U) Vs. channel SNR and the number of FBCs N_f

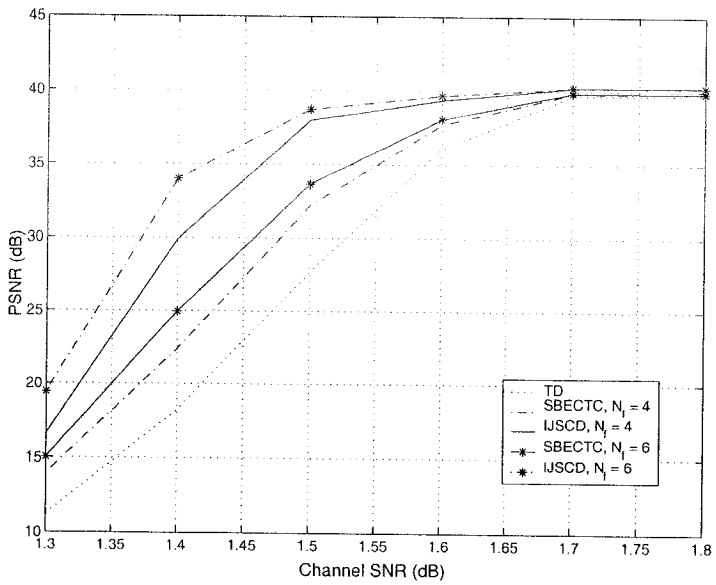

Figure 4.8: Red chrominance PSNR (V) Vs. channel SNR and the number of FBCs N_f

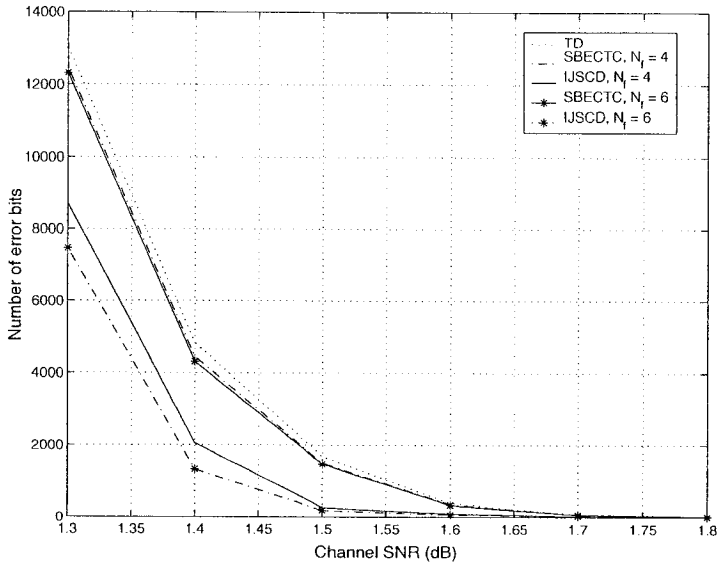

Figure 4.9: Number of error bits (NEB) Vs. channel SNR and the number of FBCs N_f

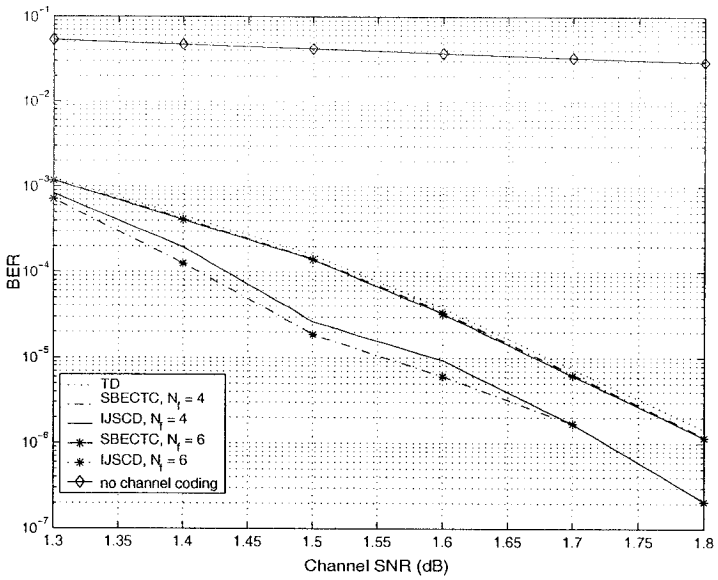

Figure 4.10: Bit error rate (BER) Vs. channel SNR and the number of FBCs N_f

For more objective performance comparison, the required channel SNR is measured for three schemes to reach identical PSNR and BER. This result is given in Table 4.6 where the data are extracted from Figure 4.6, 4.7, 4.8 and 4.10.

Table 4.6 shows that, in terms of BER, the required channel SNR of the proposed scheme using $N_f = 6$ is about 0.13 dB less than the TD and about 0.125 dB less than the SBECTC. In addition, the proposed scheme using $N_f = 6$ and $N_{VP} = 64$ can achieve 35 dB in terms of luminance PSNR (Y) at channel SNR 1.50 dB, while the TD gets the same performance at channel SNR 1.68 dB. At this point, the required channel SNR of the proposed scheme using $N_f = 6$ is 0.18 dB less than the TD.

Table 4.6: Required channel SNR to achieve the same performance when different decoding schemes employed

Performance	IJSCD with $N_f = 6$	IJSCD with $N_f = 4$	SBECTC	TD
35 dB PSNR (Y)	1.50	1.56	1.66	1.68
39 dB PSNR (U)	1.49	1.55	1.65	1.68
38.5 dB PSNR (V)	1.50	1.55	1.65	1.68
2.5×10^{-5} BER	1.47	1.51	1.625	1.63
1.0×10^{-5} BER	1.55	1.585	1.675	1.68

According to the simulation results shown in Figures 4.6, 4.7 and 4.8 for "Table-Tennis", there are two cases that the proposed scheme can achieve considerable performance improvement and the quality of the reconstructed videos is close to the available maximum PSNRs, which are presented in Table 4.7. One is using $N_f = 6$ at channel SNR 1.5 dB, the other is using $N_f = 4$ at channel SNR 1.6 dB.

Table 4.7: Maximum PSNR from the video compressor for "Table-Tennis"

PSNR (Y)	36.371584
PSNR (U)	40.837548
PSNR (V)	40.246831

2. Objective performance when Video "Football" employed

The simulations on "Football" are actually run at channel SNR 1.5 and 1.6 dB. At channel 1.5 dB, $N_f = 6$ while at channel 1.6 dB, $N_f = 6$. These two cases can provide PSNR close to the available maximum PSNRs, and at the same time, the performance is improved considerably comparing to the other two schemes, the SBECTC and the TD, as shown by the simulation results for "Table-Tennis". The other simulation conditions are the same as listed at the beginning of Section 4.4. For comparison, the available maximum PSNRs, which are calculated for the compressed but noisy free bitstream, are presented in Table 4.8.

Each set of simulation parameters are run 5 times and the simulation results shown in this part are the average values of these 5 experiments.

The simulation results at channel 1.5 and 1.6 dB are given in Table 4.9 and Table 4.10, respectively. Tables 4.9 and 4.10 indicate that the objective performance achievement using "Football" is similar to what is achieved using "Table-Tennis". That is, in these two cases, the performance improvement is considerable and the achieved PSNRs are close to the available maximum PSNRs.

Table 4.8: Maximum PSNR from the video compressor for Video "Football"

PSNR (Y)	33.786
PSNR (U)	36.64
PSNR (V)	37.93

Table 4.9: Objective performance comparison at channel SNR 1.5 dB when employing different schemes and Video "Football"

Performance	IJSCD with $N_f = 6$	SBECTC	TD
PSNR (Y)	32.788	30.421	26.293
PSNR (U)	35.861	33.87	29.761
PSNR (V)	37.144	34.869	30.586
NEB	350	1230.8	1402.4

Table 4.10: Objective performance comparison at channel SNR 1.6 dB when employing different schemes and Video "Football"

Performance	IJSCD with $N_f = 4$	SBECTC	TD
PSNR (Y)	33.283	32.472	31.254
PSNR (U)	36.31	35.643	34.494
PSNR (V)	37.701	36.802	35.693
NEB	117.2	371.6	427.2

4.4.3 Subjective Performance

The subjective performance of the proposed scheme is also evaluated for two videos, "Table-Tennis" and "Football", through simulations and the simulation conditions are exactly the same as what are used for objective performance evaluation in Section 4.4.2.

For subjective performance evaluation, the reconstructed videos at different channel SNR by three different decoding schemes, the TD, the SBECTC and the

proposed scheme, are played on the computer. The author's comments on reconstructed videos are provided. Also, for the reader's own assessment, some frames from the reconstructed videos are presented.

1. Subjective performance when Video "Table-Tennis" employed

When "Table-Tennis" is employed, the videos reconstructed by the proposed scheme are obviously better than the other two schemes. The SBECTC is better than the TD. In particular, at channel SNR 1.60 dB and $N_f = 4$, as well as channel SNR 1.50 dB and $N_f = 6$, all stripes are removed and video scene shifts are corrected when employing the proposed scheme. Moreover, no obvious defects can be found. However, when using the TD at channel SNR 1.5, there are defects in about 80% frames while at channel SNR 1.60 dB, there are defects in about 22% frames. In these two cases, the SBECTC is better than the TD, but much worse than the proposed scheme.

For the reader's own assessment, Frame 73 at channel SNR 1.50 dB and Frame 166 at channel SNR 1.60 dB from the reconstructed videos by the TD, the SBECTC with two different N_f and the proposed scheme with two different N_f are presented. In the simulations, N_f is chosen as 4 and 6, respectively. The corresponding two frames from the original video are also shown for performance comparison. All the video frames are shown in Figure from 4.11 to 4.22. The corresponding PSNRs of Frame 73 and Frame 166 reconstructed by different decoding schemes are listed in Table 4.11 and 4.12, respectively. Note that the first row in these two tables shows the PSNR values after video compression and before transmission, which represent the best achievable PSNR values. Both PSNR and picture quality indeed show that the proposed scheme IJSCD can improve the performance compared with the SBECTC and the TD.

Table 4.11: PSNR of Frame 73 of Video "Table-Tennis" at channel SNR 1.5 dB when employing different schemes and Video "Table-Tennis"

Scheme	PSNR (Y)	PSNR (U)	PSNR (V)
Video after compression	36.889660	40.156043	39.899125
TD	16.917741	18.514108	17.897322
SBECTC $N_f = 4$	17.787194	19.966472	19.250924
IJSCD $N_f = 4$	28.428199	38.887521	39.573244
SBECTC $N_f = 6$	25.239704	33.653227	34.915951
IJSCD $N_f = 6$	36.889660	40.156043	39.899125

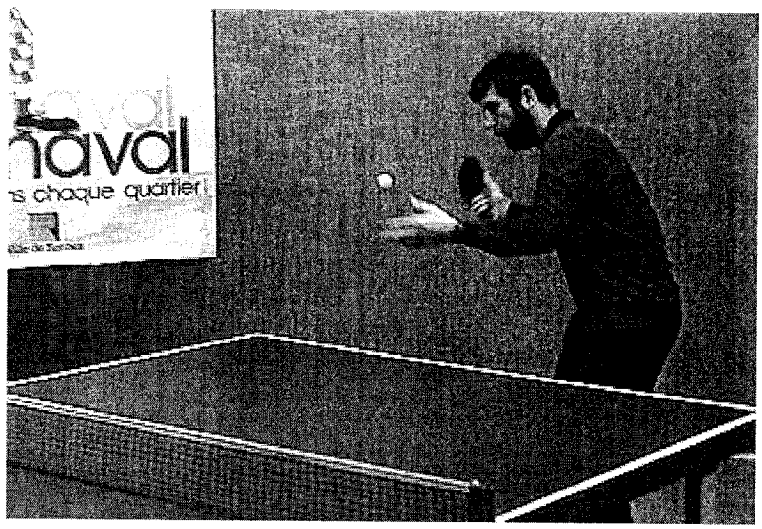

Figure 4.11: Frame 73 from the original error-free video "Table-Tennis"

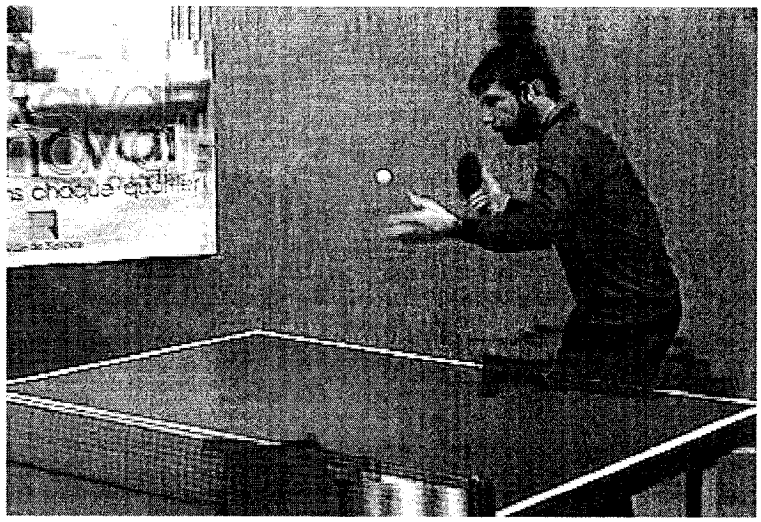

Figure 4.12: Frame 73 from the decompressed video "Table-Tennis" reconstructed by the TD

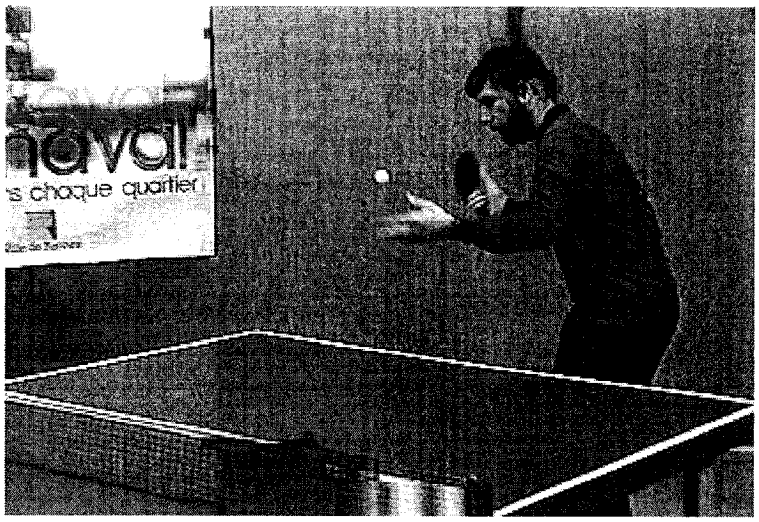

Figure 4.13: Frame 73 from the decompressed video "Table-Tennis" reconstructed by the SBECTC with $N_f = 4$

Figure 4.14: Frame 73 from the decompressed video "Table-Tennis" reconstructed by the SBECTC with $N_f = 6$

Figure 4.15: Frame 73 from the decompressed video "Table-Tennis" reconstructed by the proposed scheme, IJSCD, with $N_f = 4$

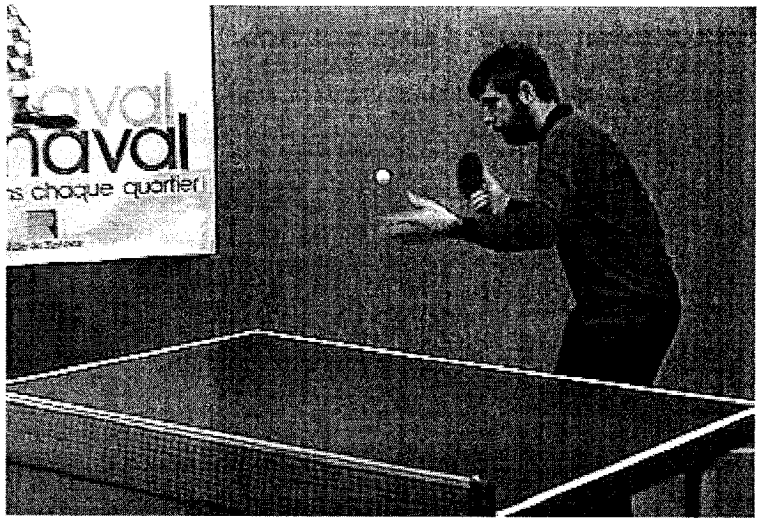

Figure 4.16: Frame 73 from the decompressed video "Table-Tennis" reconstructed by the proposed scheme, IJSCD, with $N_f = 6$

Table 4.12: PSNR of Frame 166 of Video "Table-Tennis" at channel SNR 1.6 dB when employing different schemes and Video "Table-Tennis"

Scheme	PSNR (Y)	PSNR (U)	PSNR (V)
Video after compression	33.867538	40.164476	38.658096
TD	13.510562	17.005029	16.471387
SBECTC $N_f = 4$	19.344450	22.574711	22.369891
IJSCD $N_f = 4$	33.572512	40.164476	38.658096
SBECTC $N_f = 6$	19.344450	22.574711	22.369891
IJSCD $N_f = 6$	33.867538	40.164476	38.658096

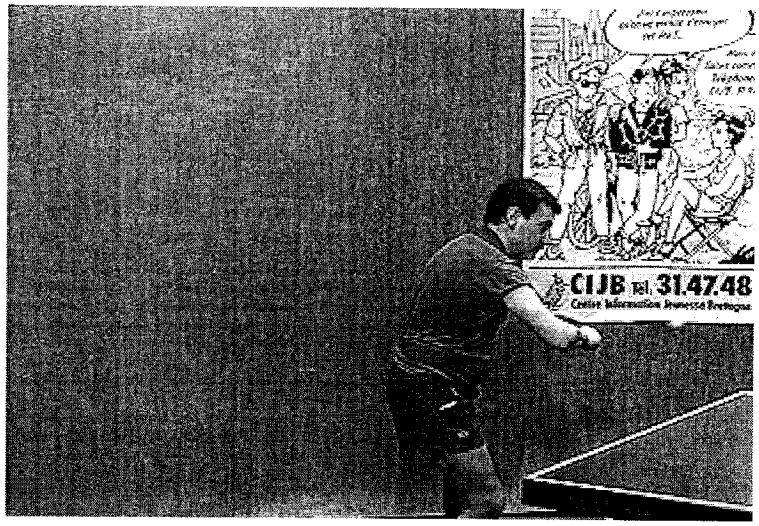

Figure 4.17: Frame 166 from the original error-free video "Table-Tennis"

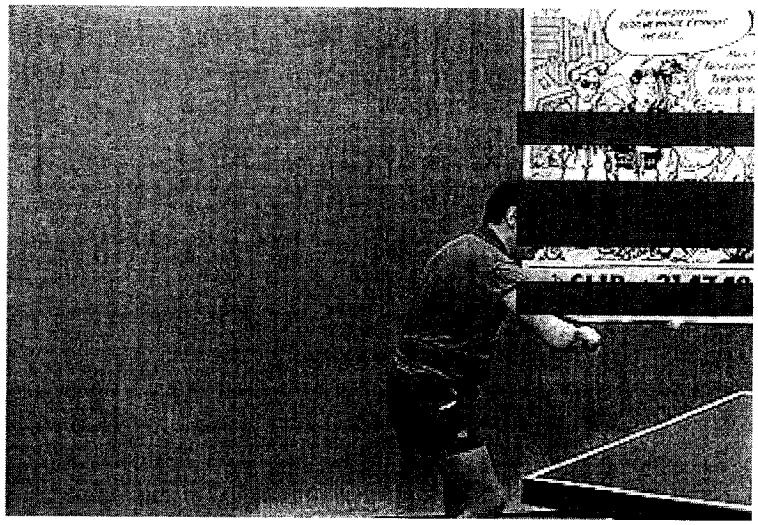

Figure 4.18: Frame 166 from the decompressed video "Table-Tennis" reconstructed by the TD

Figure 4.19: Frame 166 from the decompressed video "Table-Tennis" reconstructed by the SBECTC with $N_f = 4$

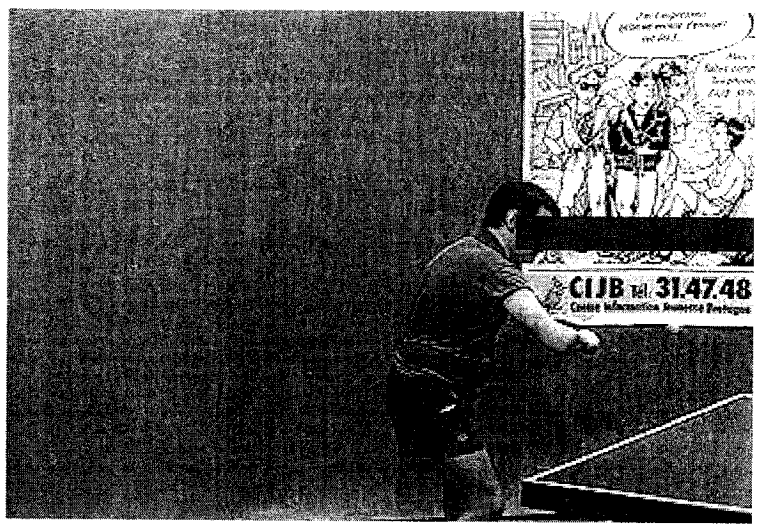

Figure 4.20: Frame 166 from the decompressed video "Table-Tennis" reconstructed by the SBECTC with $N_f = 6$

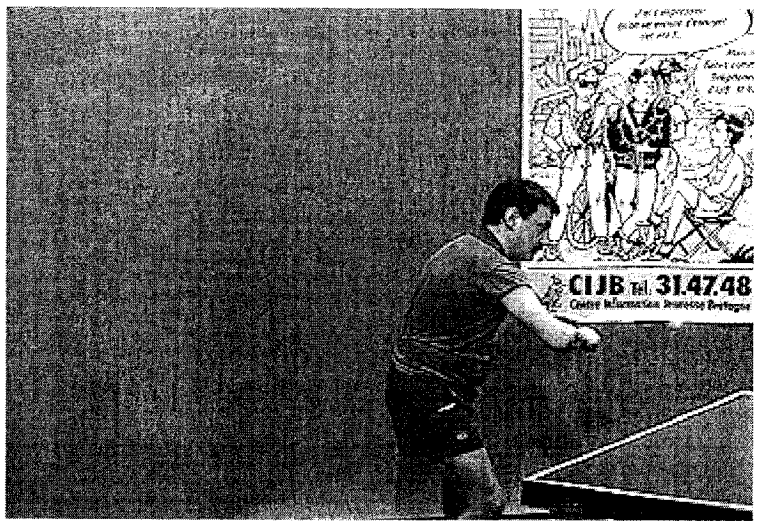

Figure 4.21: Frame 166 from the decompressed video "Table-Tennis" reconstructed by the proposed scheme, IJSCD, with $N_f = 4$

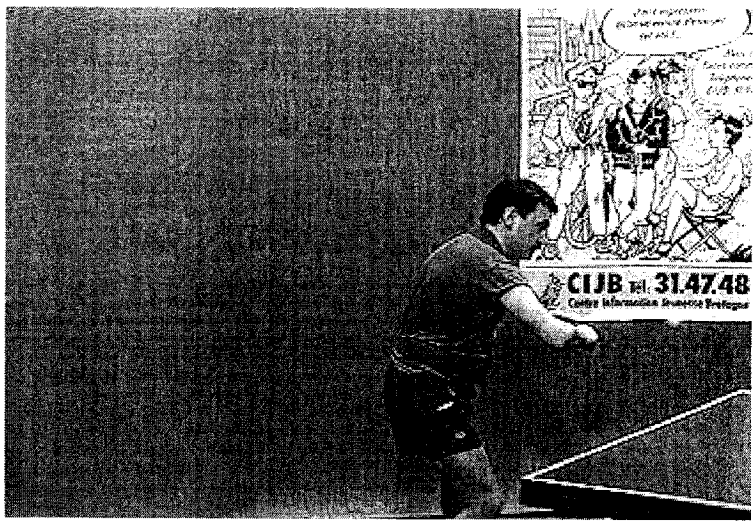

Figure 4.22: Frame 166 from the decompressed video "Table-Tennis" reconstructed by the proposed scheme, IJSCD, with $N_f = 6$

From the simulation results shown in Tables 4.11 and 4.12, it can be seen that the proposed scheme with both $N_f = 6$ at channel SNR 1.50 dB and $N_f = 4$ at channel SNR 1.60 dB achieves the available maximum PSNR. Although that is the case for most frames, it's possible that some errors can't be corrected by the proposed scheme. The example is Frame 90 at channel SNR 1.60 dB and $N_f = 4$. Table 4.13 shows PSNR values recovered by three different schemes. The original error free frame and the recovered frames are shown in Figures 4.23 to 4.26.

Table 4.13: PSNR of Frame 90 of Video "Table-Tennis" at channel SNR 1.6 dB when employing different schemes and Video "Table-Tennis"

Scheme	PSNR (Y)	PSNR (U)	PSNR (V)
Video after compression	34.089445	37.800852	37.094272
TD	21.250589	21.983037	19.326565
SBECTC $N_f = 4$	22.492425	24.160063	21.362955
IJSCD $N_f = 4$	22.584420	24.160063	21.362955

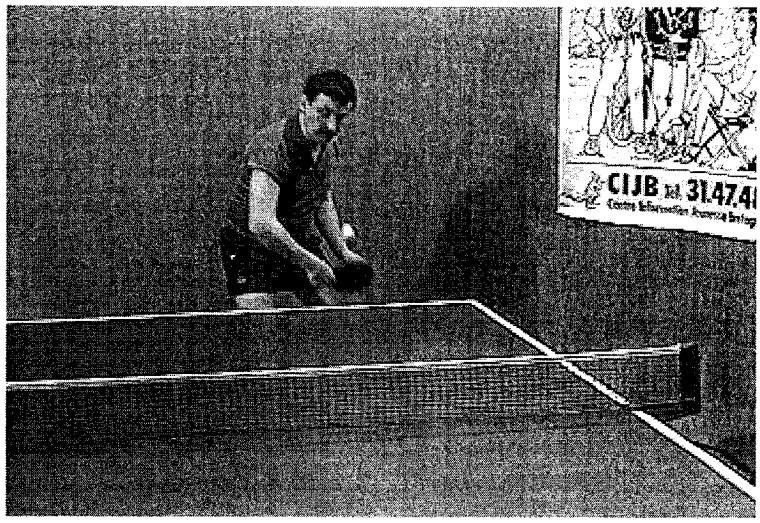

Figure 4.23: Frame 90 from the original error-free video "Table-Tennis"

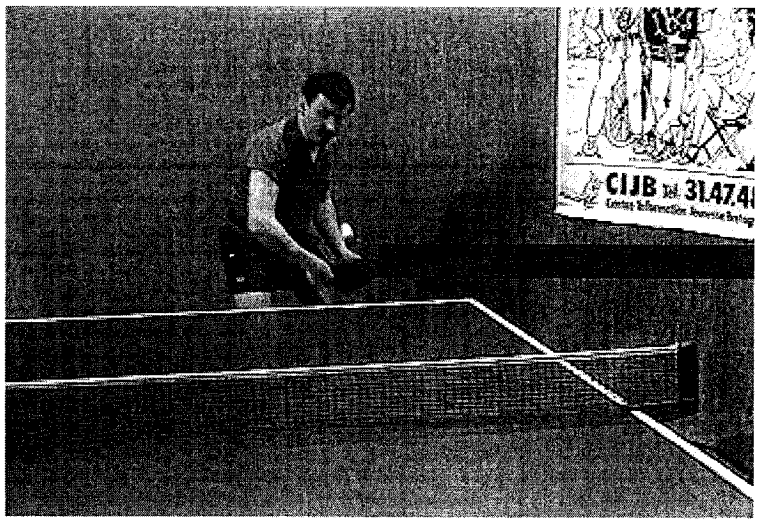

Figure 4.24: Frame 90 from the decompressed video "Table-Tennis" reconstructed by the TD

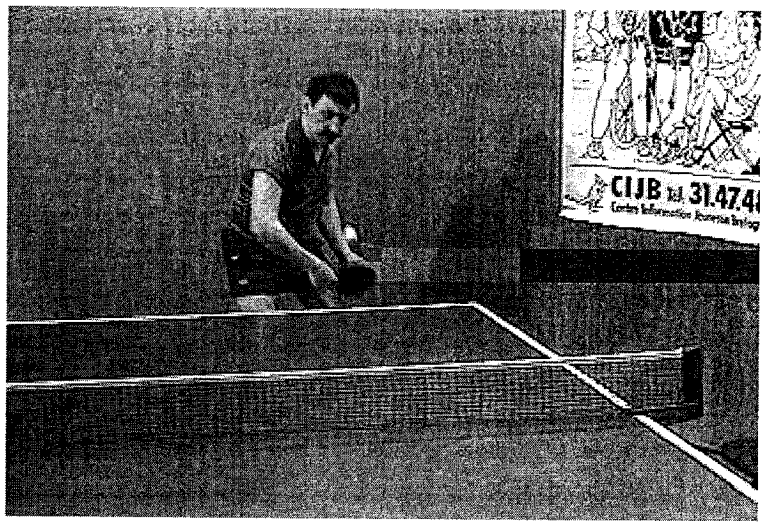

Figure 4.25: Frame 90 from the decompressed video "Table-Tennis" reconstructed by the SBECTC with $N_f = 4$

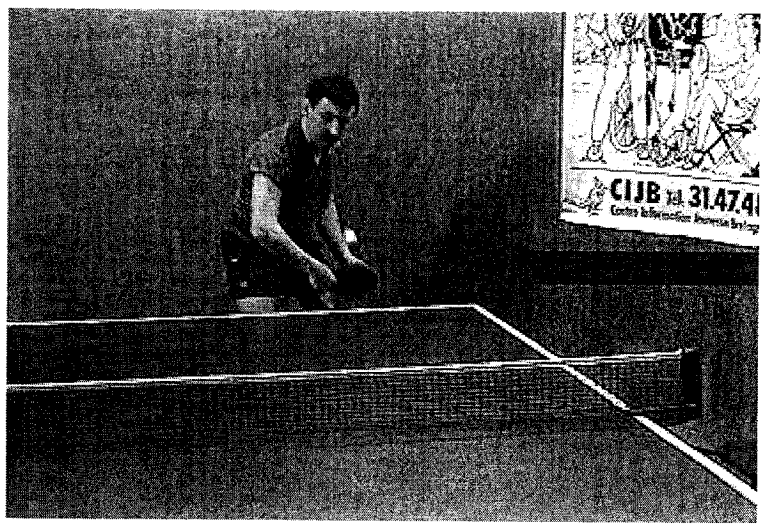

Figure 4.26: Frame 90 from the decompressed video "Table-Tennis" reconstructed by the proposed scheme, IJSCD, with $N_f = 4$

2. Subjective performance when Video "Football" employed

When "Football" is employed, the simulations are implemented at channel SNR 1.60 dB and $N_f = 4$, as well as channel SNR 1.50 dB and $N_f = 6$. The quality of the videos reconstructed by the proposed scheme are also obviously better than the other two schemes, and the SBECTC is better than the TD.

For the reader's own assessment, Frame 89 at channel SNR 1.50 dB and Frame 31 at channel SNR 1.60 dB from the reconstructed videos by the TD, the SBECTC and the proposed scheme are presented. The corresponding two frames from the original video are also shown for performance comparison. All the video frames are shown in Figures from 4.27 to 4.34.

The PSNRs of Frame 89 and Frame 31 reconstructed by different decoding schemes are listed in Table 4.14 and 4.15, respectively. Both PSNR and picture quality indeed show that the proposed scheme achieves much better performance than the SBECTC and the TD.

Table 4.14: PSNR of Frame 89 of Video "Football" at channel SNR 1.5 dB when employing different schemes

Scheme	PSNR (Y)	PSNR (U)	PSNR (V)
Video after compression	30.592183	35.060759	36.873958
TD	16.334625	17.750338	15.482997
SBECTC $N_f = 6$	26.491166	28.047710	26.301719
IJSCD $N_f = 6$	30.592183	35.060759	36.873958

Table 4.15: PSNR of Frame 31 of Video "Football" at channel SNR 1.6 dB when employing different schemes

Scheme	PSNR (Y)	PSNR (U)	PSNR (V)
Video after compression	31.821580	35.297888	36.414425
TD	14.291031	15.633579	12.891659
SBECTC $N_f = 4$	20.325286	22.005553	19.672725
IJSCD $N_f = 4$	31.821580	35.297888	36.414425

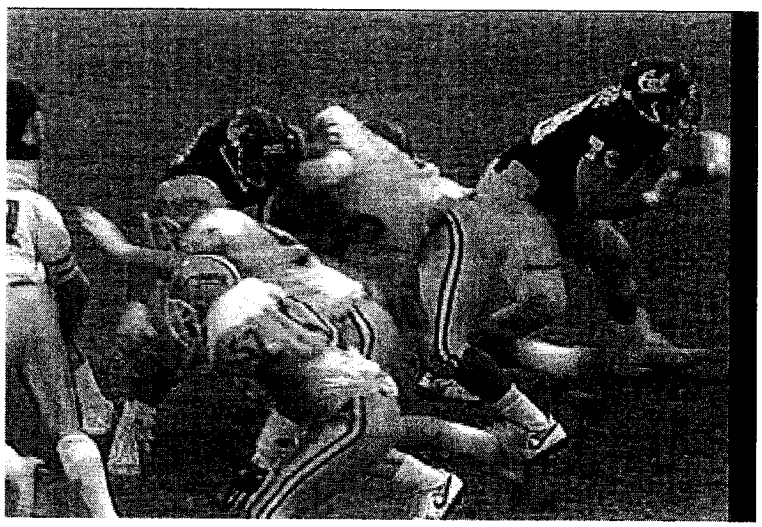

Figure 4.27: Frame 89 from the original error-free video "Football"

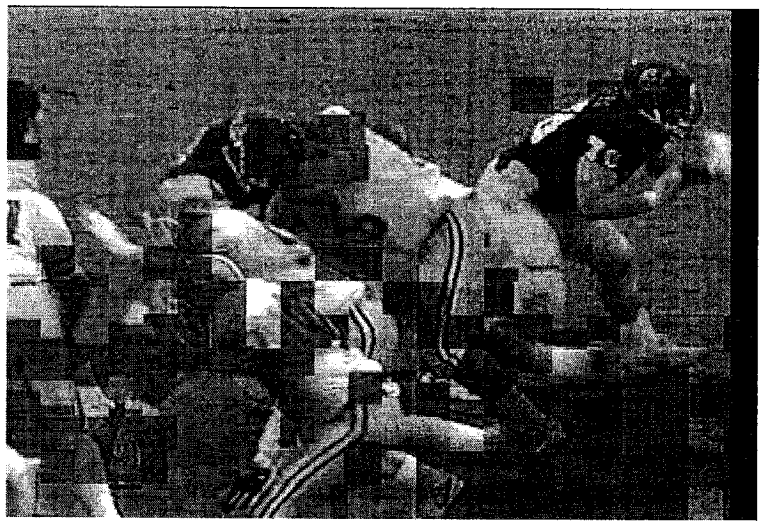

Figure 4.28: Frame 89 from the decompressed video "Football" reconstructed by the TD

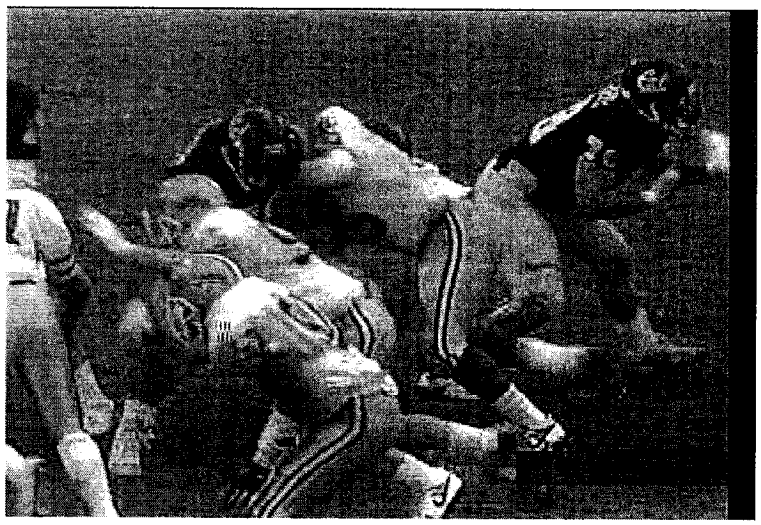

Figure 4.29: Frame 89 from the decompressed video "Football" reconstructed by the SBECTC with $N_f = 6$

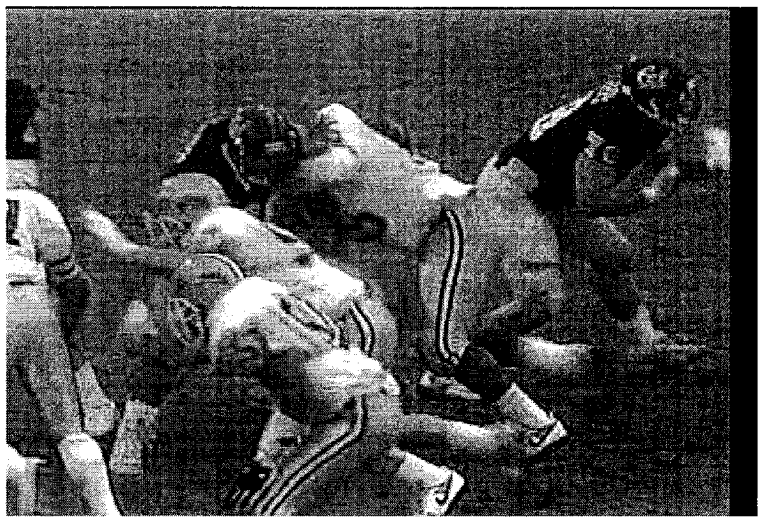

Figure 4.30: Frame 89 from the decompressed video "Football" reconstructed by the proposed scheme, IJSCD, with $N_f = 6$

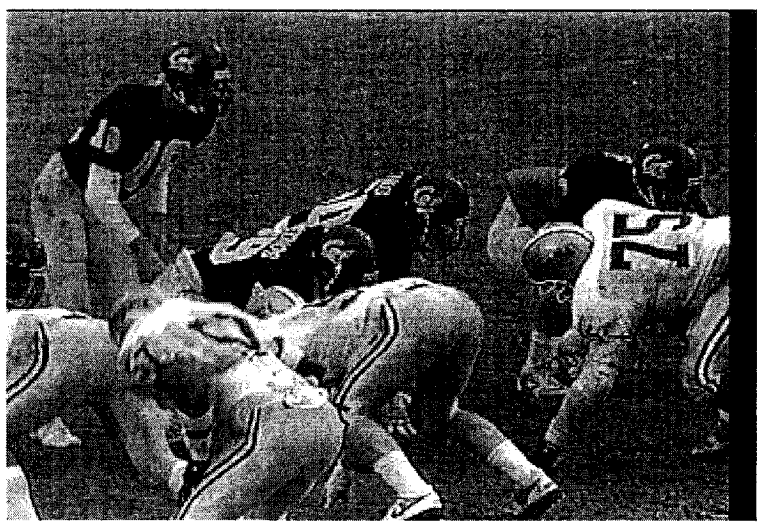

Figure 4.31: Frame 31 from the original error-free video "Football"

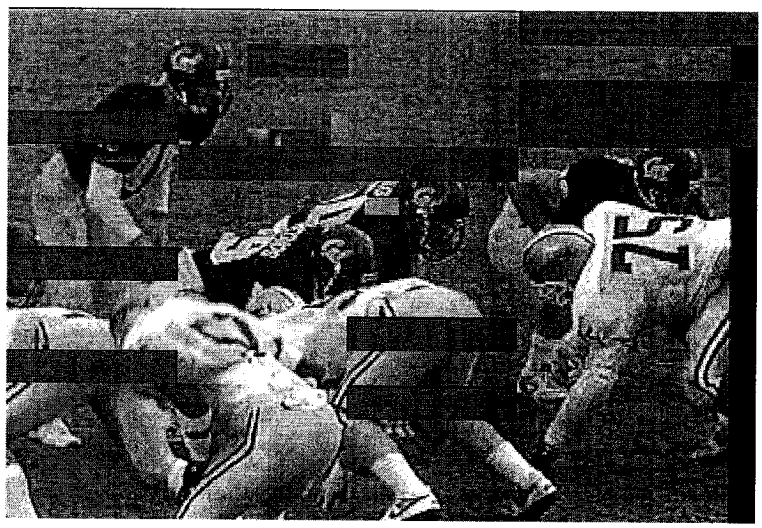

Figure 4.32: Frame 31 from the decompressed video "Football" reconstructed by the TD

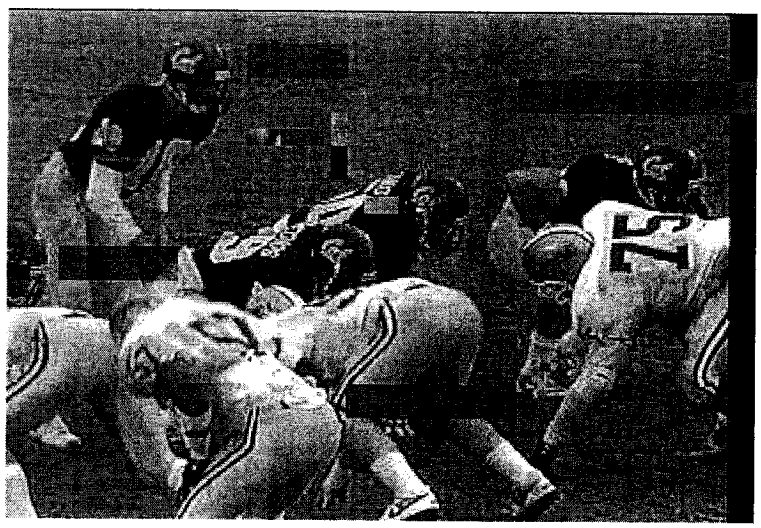

Figure 4.33: Frame 31 from the decompressed video "Football" reconstructed by the SBECTC with $N_f = 4$

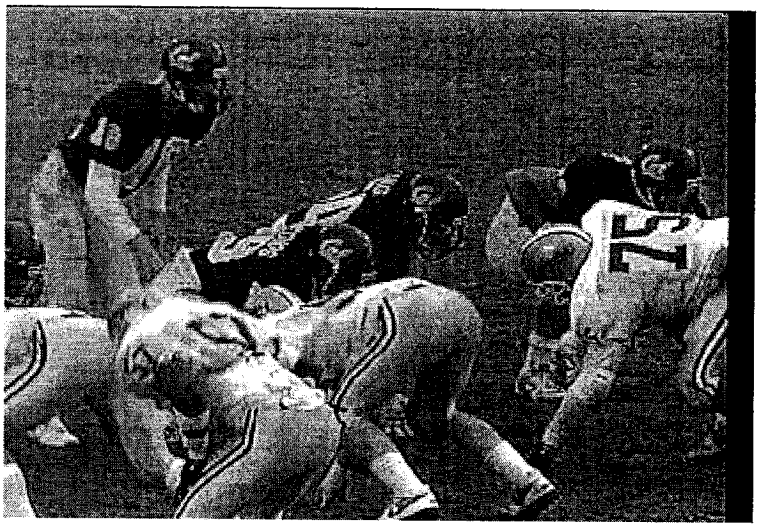

Figure 4.34: Frame 31 from the decompressed video "Football" reconstructed by the proposed scheme, IJSCD, with $N_f = 4$

4.4.4 Computational Time Complexity

Only computational time complexity is consider in this work. The computational time complexity of the proposed scheme IJSCD is evaluated by comparing it with those of the other two schemes, the SBECTC and the TD. The complexity evaluation is carried out through simulations and the simulation conditions are the same as what are used for objective and subjective performance evaluation for "Table-Tennis".

There are two components of complexity which are common for all three schemes. One common complexity is spent on the turbo decoding process. The time-unit complexity of MAP decoding algorithm of a single RSC code $(n, 1, m)$ has been presented in Section 2.2.5. n and m of the RSC code used in this thesis, which are introduced in Section 3.3.1, are equal to 2 and 4, respectively. It's known that the complexity of MAP turbo decoding is high.

The other common complexity comes from the video decompressing process. Unlike the complexity of turbo decoding process, the complexity of video decompression is hard to quantify since the Video Decompressor is quite a complicated system. Previous work introduces a method of complexity evaluation by counting

the number of extracted bits and slices(VP in MPEG-4 terminology) [35, 38]. In this thesis, the number of extracted bits and VPs are also used to measure the complexity in video decompression. It should be noted that the number of extracted bits and extracted VPs are two different measures for evaluating one thing, i.e., the complexity of video decompression. Since many syntax errors can be caught before the end of a VP, it seems that the complexity is measured more accurately by the number of extracted bits than that of extracted VPs. However, both measures are still used in later complexity evaluation.

In addition to the common complexity, each scheme compared may have its own requirement on complexity. The complexity of the TD is simple and is made up of the two common requirements discussed above. Besides the complexity on turbo decoding and video decompression, the SBECTC has extra complexity for constructing and decompressing multiple VP candidates at the last iteration, which is done by the MVPCG module. The proposed scheme, IJSCD, requires extra complexity at each iteration for constructing and decompressing multiple VP candidates as well as modifying $L_{21}^{e}(u_k)$ of the bits, which is done by the MM, in the best VP candidates which has been considered as error-free by the Video Decompressor.

The complexity on constructing multiple VP candidates can be divided into two parts:

- Choosing N_f bits with the smallest $|L_2(u_k)|$ in a video packet

  When the number of bits in a VP is denoted as x and Quicksort algorithm is employed for this job, the complexity is $x \log_2 x$, which is used for comparison between two real numbers [39]. In the simulation while counting the number of extracted bits, the number of bits in the VP (x) is also counted.

- Choosing N_{VP} VP candidates from the possible 2^{N_f} combinations

  This job is similar to the above job, except that 2^{N_f} is much smaller than the number of bits in a VP (x). In the simulations, the maximum number of N_f

is 6. Hence, the complexity of this job can be neglected.

Modification on $L_{21}^{e}(u_k)$ of the bits in a VP is an addition operation as discussed in 4.2. When a VP is considered as error free, modification is required no matter that this VP is the original one from the turbo decoder or from the multiple VP candidates. If the number of bits in the VP is x, the complexity is just x additions. The number of bits is obtained when counting the extracted bits.

To compare the complexity of three schemes, some simulations have been implemented based on the above discussions. The simulations are run at channel 1.60 dB. N_{VP} and N_f are chosen as 16 and 4, respectively. The other simulation conditions are the same as in Section 4.4. The first experiment in this case is used for complexity comparison, not with 5 experiments as what have done before.

Table 4.16 lists the complexity of three different schemes after 15 iterations. From this table, it can be observed that the total complexity on addition, multiplication and exponential operation of three schemes are very close due to the huge complexity on the turbo decoding process. The comparison operation can be neglected since it's in the order of 10^8 while the addition, multiplication and exponential operation are in the order of 10^{10}.

In terms of the complexity on the extracted bits/VPs, the proposed scheme is much higher than the other two schemes. At channel SNR 1.6 dB and $N_f = 4$, the complexity of the proposed scheme, in terms of the extracted bits, is in the order of 10^8 and about 43.46 times that of the TD. The complexity of the SBECTC is about 2.155 times that of the TD. It should be noted that the number of extracted bits has a rough relation to how many times a video is decompressed. However, the video decompression operations on checking syntax errors as in the proposed scheme are much less than those of full decompression of videos.

In addition, the complexity is also compared in terms of run time after 15-iteration of turbo decoding. The run time of the proposed scheme is about 1.5 times that of the TD, and the SBECTC is about 1.05 times that of the TD. However, the

extra complexity results in improved performance, in terms of PSNR and NEB, as shown in Table 4.16.

Table 4.16: Complexity comparison at channel SNR 1.6 dB after 15 iterations ($N_m = 20$, $N_{VP} = 16$, $N_f = 4$, "Table-Tennis".)

	TD	SBECTC	IJSCD
Extracted bits	10,329,000	22,261,593	448,973,070
Extracted VP	10,860	12,120	504,282
Addition (Turbo decoding, Modification)	3.3712×10^{10}	3.3712×10^{10}	3.3722×10^{10}
Multiplication (Turbo decoding)	4.9809×10^{10}	4.9809×10^{10}	4.9809×10^{10}
Exponential operation e^x (Turbo decoding)	1.9438×10^{10}	1.9438×10^{10}	1.9438×10^{10}
Comparison (MVPCG)	0	3,505,679	337,141,395
PSNR (Y)	33.306371	34.001393	36.332587
NEB	363	345	14
Iteration time	15	15	15

For complexity comparison, the complexity is also calculated when the proposed scheme and the SBECTC, after a certain iterations, achieve similar or better performance than 15 iterations of the TD. The results are presented in Table 4.17.

Table 4.17 shows that the proposed scheme achieves similar performance as both the TD and the SBECTC while requiring much lower complexity on addition, multiplication and exponential operations since the required iteration time is decreased dramatically. These operations are still in the order of 10^{10}. The total addition, multiplication and exponential operations required by the proposed scheme is about 53 percent of those required by the TD. These operations required by the SBECTC is about 93 percent of those required by the TD.

In terms of the number of the extracted bits, the bits extracted by the SBECTC are about 2.18 times those extracted by the TD. The bits extracted by the proposed scheme are about 28.99 times of those extracted by the TD.

From the comparison above, the complexity of the proposed scheme, in terms of extracted bits/VPs is much higher than the SBECTC and TD. However, the point is that the objective and subjective performance achieved by the proposed scheme as shown in Section 4.4.2 and 4.4.3 can't be reached by either the SBECTC or the TD.

Table 4.17: Complexity evaluation at channel SNR 1.6 dB when achieving similar performance in terms of PSNR and NEB($N_m = 20$, $N_{VP} = 16$, $N_f = 4$, "Table-Tennis".)

	TD	SBECTC	IJSCD
Extracted bits	10,329,000	22,473,407	299,501,103
Extracted VP	10,860	12,289	424,714
Addition (Turbo decoding, Modification)	3.3712×10^{10}	3.1465×10^{10}	1.8026×10^{10}
Multiplication (Turbo decoding)	4.9809×10^{10}	4.6489×10^{10}	2.6565×10^{10}
Exponential operation e^x (Turbo decoding)	1.9438×10^{10}	1.8142×10^{10}	1.0367×10^{10}
Comparison (MVPCG)	0	4,239,768	324,254,448
PSNR (Y)	33.306371	33.534134	34.830597
NEB	363	368	366
Iteration time	15	14	8

4.5 Summary

This chapter has discussed the proposed scheme, iterative joint source and channel decoding (IJSCD) in detail in Section 4.1. The proposed scheme provides the interactivity between the channel decoder(turbo decoder) and the source decoder. The soft output of the turbo decoder is iteratively used for syntax based error concealment in the source decoder, which is done through the Syntax Checker and MVPCG in Figure 4.1. The extracted syntax/semantic information from the source decoder is iteratively fed back to the turbo decoder to improve the error performance of turbo codes through the MM in Figure 4.1.

The performance of the proposed scheme is evaluated from the viewpoint of objective performance, subjective performance and complexity in Section 4.4. The simulation results show that the proposed scheme can achieve better performance than the other two schemes, the TD and the SBECTC, in terms of PSNR, BER and subjective video quality. In particular, the proposed scheme with 4 FBCs at channel SNR 1.5 dB and with 6 FBCs at channel SNR 1.5 dB can achieve considerable performance improvement and reconstruct the videos with quality close to the available maximum PSNR. Moreover, the required channel SNR by the proposed scheme is about 0.18 dB less than the other two schemes while all three schemes reach the same PSNR. However, the complexity of the proposed scheme is higher than the other two schemes.

It should be noted that the parameter values employed in the simulations can be refined through further empirical investigations.

Chapter 5

Conclusion and Future Work

5.1 Contributions

This thesis proposes an iterative joint source and channel decoding scheme for robust MPEG-4 video transmission. The proposed scheme uses turbo codes to provide error protection for MPEG-4 compressed bitstreams. In the decoding process, on one hand, the soft output from the turbo decoder is served as a reliability measure of the corresponding hard decision of each received bit and used for syntax based error concealment in the source decoder. On the other hand, the extracted syntax/semantic information from the source decoder is fed back to the turbo decoder by modifying the extrinsic information generated by the MAP decoder 2, one of the two constituent MAP decoders inside the turbo decoder. In this way, the interactivity between the source decoder and channel decoder is constructed and this interactivity works iteratively along with the turbo decoding process. These are the main contributions of this work.

5.2 Conclusions

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

- As expected, the soft output $L_i(u_k)$ associated with a received bit from a turbo decoder can be used to measure the reliability of the corresponding hard decision made by the turbo decoder. In general, correctly decoded bits have large $|L_i(u_k)|$ values with high possibility.

- Syntax/semantic errors in a compressed bitstream results in the most serious degradation in terms of PSNR and subjective video quality since they often make a VP un-decodable.

- A compressed bitstream is sensitive to channel induced errors due to error propagation. On the other hand, this sensitivity can be an advantage for Syntax Based Error Concealment(SBEC). That is, if a few erroneous bits in a

frame are corrected by SBEC, the picture errors in several frames referred to this frame can also be corrected.

- The residual redundancy remaining in compressed video bitstreams can be easily accessed if there are only a few bit errors in one VP.

- The information from the channel is useful for source decoding and the residual information from the source is also helpful for channel decoding. A joint source and channel decoding scheme works better than separate decoding schemes, in terms of the quality of reconstructed sources and bit error rate.

5.3 Future Work

The residual information considered in this thesis is only the syntactic/semantic correctness of a VP candidate, other residual information such as the discontinuity measure introduced in Section 3.1 can be employed in future work. It can be expected that more information from the source can be more helpful for channel decoding, and error concealing in post-processing of videos.

Some work can be done to optimize the proposed scheme aiming to reduce the complexity. For example, the feedback scheme may begin to work after 2 or 3 iterations of turbo decoding. Also, the number of VP candidates doesn't need to be fixed. Instead, it can be associated to the results of last iteration.

The feedback scheme, which sends the information from the source to the turbo decoder, is based on a simple principle in this thesis, that is, if the hard decision of a bit made by the source decoder is in agreement with the one made by the turbo decoder, the feedback scheme increases the absolute value of its corresponding soft output from the turbo decoder. Otherwise, the feedback scheme decreases it. How much the absolute value of the soft output from the turbo decoder can be increased/decreased is determined by experiments. In future work, some theoretical

work on the feedback scheme needs to be developed and the ultimate goal is to convert the extracted source information to soft values at bit level in a mathematical way.

REFERENCES

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

[2] P. Symes, *Video Compression Demystified.* New York: Mcgraw-Hill, 1st ed., 2001.

[3] J. Watkinson, *The* MPEG *Handbook.* 225 Wildwood Avenue, Woburn, MA 01801-2041: Focal Press, 1st ed., 2001.

[4] I. Moccagatta, S. Soudagar, J. Liang, and H. Chen, "Error-resilient coding in JPEG-2000 and MPEG-4," *IEEE journal on selected areas in communications*, vol. 18, pp. 899–914, June 2000.

[5] B. Sklar, *Digital Communications Fundamentals and Applications.* Upper Saddle River,NJ 07458: Prentice Hall, 2nd ed., 2001.

[6] S. Lin and D. J. C. Jr., *Error control coding Fundamentals and Applications.* Englewood Cliffs, New Jersey 07632: Prentice Hall, 1st ed., 1983.

[7] I. E. G. Richardson, *H.264 and* MPEG-*4 video compression.* The Atrium, Sothern Gate, Chichester, West Sussex PO19 8SQ, England: John Wiley & Sons Ltd., 1st ed., 2003.

[8] *The official MPEG committee website, http://www.chiariglione.org/mpeg/.*

[9] "Information technology - generic coding of audio-visual objects," *ISO/IEC 14496-2*, 1998.

[10] F. Pereira and T. Ebrahimi, *The* MPEG-*4 Book*. Upper Saddle River, NJ 07458: Prentice Hall PTR, 1st ed., 2002.

[11] R. Talluri, "Error-resilient video coding in the ISO MPEG-4 standard," *IEEE Communications Magazine*, pp. 112–119, June 1998.

[12] A. E. Walsh and M. Bourges-Sévenier, MPEG-*4 Jump-Start*. Upper Saddle River, NJ 07458: Prentice Hall PTR, 1st ed., 2002.

[13] W. E. Lynch, V. Papadakis, R. Krishnamurthy, and T. Le-Ngoc, "Syntax based error concealment," *Signal Processing: Image Communication*, vol. 16, pp. 827–835, June 2001.

[14] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near shannon limit error-correcting coding and decoding: turbo codes," *Proc. 1993 Inter. Conf. Commun.*, pp. 1064–1070, 1993.

[15] S. Benedetto and G. Montorsi, "Unveiling turbo codes: some results on parallel concatenated coding schemes," *IEEE Transactions on Information Theory*, vol. 42, pp. 409–428, March 1996.

[16] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo codes," *IEEE Transactions on Communications*, vol. 44, pp. 1261–1271, October 1996.

[17] S. Benedetto and G. Montorsi, "Design of parallel concatenated convolutional codes," *IEEE Transactions on Communications*, vol. 44, pp. 592–600, May 1996.

[18] B. Vucetic and J. Yuan, *Turbo codes: principles and applications.* 101 Philip Drive, Assinippi Park, Norwell, Massachusetts 02061: Kluwer Academic Publishers, 1st ed., 2000.

[19] L. R. Bahl, J. Cocke, F. Jelinek, and J. Raviv, "Optimal decoding of linear codes for minimizing symbol error rate," *IEEE Transactions on Information Theory*, pp. 284–287, March 1974.

[20] J. Hagenauer, E. Offer, and L. Papke, "Iterative decoding of binary block and convolutional codes," *IEEE Transactions on Information Theory*, vol. 42, pp. 429–445, March 1996.

[21] W. E. Ryan, "A turbo code tutorial," *http://www.ece.arizona.edu/ ryan/.*

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

[23] C. W. Chen, P. Cosman, N. Kingsbury, J. Liang, and J. W. Modestino, "Error-resilient image and video transmission," *IEEE journal on selected areas in communications*, vol. 18, pp. 809–813, June 2000.

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

[25] S. Valente, C. Dufour, F. Grolière, and D. Snook, "An efficient error concealment implementation for MPEG-4 video streams," *IEEE transactions on consumer electronics*, vol. 47, pp. 568–578, August 2001.

[26] J. Hagenauer, "Source-controlled channel decoding," *IEEE Transactions on Communications*, vol. 43, pp. 2449–2457, September 1995.

[27] W. E. Lynch and T. Le-Ngoc, "Syntax and discontinuity based error concealment for compressed video in a packet environment," *ITCC-01*, pp. 258–262, 2001.

[28] 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, pp. 971–980, October 2003.

[29] W. Xu, J. Hagenauer, and J. Hollmann, "Joint source-channel decoding using the residue redundancy in compressed images," *Data Compression Conference*, pp. 142–148, 1996.

[30] J. Kliewer and N. Görtz, "Iterative source-channel decoding for robust image transmission," *Acoustic, Speech, and Signal Processing, 2002 IEEE conference*, vol. 3, pp. 2173–2176, 2002.

[31] Z. Peng, Y.-F. Huang, D. J. Costello, and R. L. Stevenson, "Joint channel and source decoding for vector quantized images using turbo codes," *ISCAS-98*, 1998.

[32] Z. Peng, Y.-F. Huang, D. J. Costello, and R. L. Stevenson, "Joint decoding of turbo codes for subband coded image," *ICIP 98*, vol. 1, pp. 329–333, October 1998.

[33] S. Aign, "Error concealment enhancement by using the reliability outputs of a sova in MPEG-2 video decoder," *ISSSE 95*, pp. 25–27, 1995.

[34] M. Bystrom, S. Kaiser, and A. Kopansky, "Soft source decoding with applications," *IEEE transactions on circuits and systems for video technology*, vol. 11, pp. 1108–1120, 2001.

[35] W. E. Lynch, V. Papadakis, R. Krishnamurthy, and T. Le-Ngoc, "Syntax and discontinuity based error concealment," *ISCAS*, pp. IV–235 – 268, May 1999.

[36] O. Y. Takeshita, O. M. Collins, P. C. Massey, and J. Daniel J. Costello, "On the frame-error rate of concatenated turbo codes," *IEEE transactions on communications*, vol. 49, pp. 602–608, April 2001.

[37] A. C. Reid, T. A. Gulliver, and D. P. Taylor, "Convergence and errors in turbo-decoding," *IEEE transactions on communications*, vol. 49, pp. 2045–2051, December 2001.

[38] V. Papadakis, W. E. Lynch, and T. Le-Ngoc, "Syntax based error concealment," *ISCAS*, pp. IV–265 – 268, June 1998.

[39] W. H. Press, S. A. Teokoisky, W. T. Vetterling, and B. T. Flannery, *Numerical Recipes in C.* The Pitt building, Trumpington Street, Cambridge, CB2 1RP: Cambridge University Press, 2nd ed., 2002.

Appendix A

Programs

A.1 Smallest Combination Algorithm

```
#include <string.h>
#include <assert.h>
#include <stdio.h>


struct combination_data {
  Double  num;      /* logarithm difference summation  */
  Int     nElem;    /* Number of indexes to produce the above number */
  Int     *elem;    /* Indexes to produce the above number */
};
typedef struct combination_data CombinationData;



/* Comment the following #define to remove the debug message */
#define __SMALLEST_COMBINATION_DEBUG__

void smallestCombination(double *num, int numSize,
CombinationData* result, int resultSize){

  int i, j, k, l, m, lastResult, lastResultIndex, pos,
    newNum, hasCommonElement, isRepeated;
  int *e1, *e2, *e;
  CombinationData newComb;

  /* Precondition */
  assert(numSize>0 && resultSize>0 && num!=0
    && result!=0 );/* numSize>=resultSize */

  /* Allocate initial memory to store the indexes */
  for (i=0; i<resultSize; i++){
    result[i].nElem = 1;
    result[i].elem  = (int*)malloc(sizeof(int));
  }


  /* Initialize result with smallest single "resultSize" numbers */
  lastResult=0;   lastResultIndex=resultSize-1;
  result[0].nElem   = 1;
  result[0].num     = num[0];
  result[0].elem    = (int*)malloc(sizeof(int));
  result[0].elem[0] = 0;
  for (i=1; i<numSize; i++){
    assert(num[i] >= 0);
            /* Precondition: All numbers must be non-negative */

    /* Find the insertion position "pos" */
    if (num[i]>=result[lastResult].num) {
```

```
    if (lastResult<lastResultIndex){
      /* Set insertion to the 1 past last location (append) */
      lastResult++;
      pos = lastResult;
    }
    else continue;
      /* No room => Current element must not be in the result set */
  }
  else {
    for (pos=0; pos<=lastResult; pos++)
      if (num[i] < result[pos].num)
        break;

    if (lastResult<lastResultIndex) lastResult++;
    else free(result[lastResult].elem);
    for (j=lastResult; j>pos; j--) result[j]=result[j-1];
  }

  /* Insert num[i] to "pos" */
  result[pos].nElem   = 1;
  result[pos].num     = num[i];
  result[pos].elem    = (int*)malloc(sizeof(int));
  result[pos].elem[0] = i;
}

#ifdef __SMALLEST_COMBINATION_DEBUG__
printf("-------------------------------------------------------\n");
printf("Index:  Number: Element Index\n");
for (k=0; k<=lastResult; k++){
  printf("%6d%8.2f:",k,result[k].num);
  e = result[k].elem;
  for (l=0; l<result[k].nElem; l++)
    printf(" %3d",e[l]);
  printf("\n");
}
printf("\n");
#endif

/* Main loop to find out the smallest combinations */
for (i=0; i<lastResult; i++){

  /* Fast termination condition */
  if (result[i].num+result[i+1].num>=
      result[lastResultIndex].num /* New number is too big */
      && lastResult==lastResultIndex /* No room to append */ ){
    #ifdef __SMALLEST_COMBINATION_DEBUG__
    printf("i = %2d: Fast termination (No need to continue) !\n", i);
    #endif
    return;
  }
```

```
for (j=i+1; j<=lastResult; j++){
  #ifdef __SMALLEST_COMBINATION_DEBUG__
  printf("i = %2d, j = %2d\n",i,j);
  #endif

  newNum = result[i].num + result[j].num;
  if (newNum>=result[lastResultIndex].num &&
    lastResult==lastResultIndex) {
    #ifdef __SMALLEST_COMBINATION_DEBUG__
    printf("  Skipped: Combination>
        Last result and No room to append.\n");
    #endif
    break;
  }

  /* Check if there are common elements
  among result[i].elem and result[j].elem */
  hasCommonElement=0;   e1=result[i].elem;    e2=result[j].elem;
  for (k=0; k<result[i].nElem; k++)
    for (l=0; l<result[j].nElem; l++)
      if (e1[k]==e2[l]) {
        hasCommonElement=1; break;
      }
  if (hasCommonElement!=0) {
    #ifdef __SMALLEST_COMBINATION_DEBUG__
    printf("  Skipped: Has common elements.\n");
    #endif
    continue; /* Not a valid combination (has repeated element) */
  }

  /* Create new combination (not inserted yet) */
  newComb.num   = newNum;
  newComb.nElem = result[i].nElem + result[j].nElem;
  newComb.elem  = (int*)malloc(newComb.nElem*sizeof(int));
  k=0;  e=newComb.elem;   e1=result[i].elem;  e2=result[j].elem;
  for (k=0,l=0,m=0; k<newComb.nElem; k++){
    /* Create a sorted merged list */
    if (e1[l]<e2[m]){
      e[k] = e1[l];   l++;
      if (l==result[i].nElem) break;
    }
    else {
      e[k] = e2[m];   m++;
      if (m==result[j].nElem) break;
    }
  }
  if (l<result[i].nElem){
    for (k++; k<newComb.nElem; k++, l++) e[k]=e1[l];
  }
```

```
else if (m<result[j].nElem){
  for (k++; k<newComb.nElem; k++, m++) e[k]=e2[m];
}

#ifdef __SMALLEST_COMBINATION_DEBUG__
printf("New Combination: %8.2f: ",newComb.num);
for (k=0; k<newComb.nElem; k++) printf("%2d ",e[k]);
#endif

/* Check if the new combination is
  already presented in the result */
isRepeated = 0;
for (k=j; k<=lastResult && isRepeated==0; k++){
  if (newComb.num==result[k].num &&
      newComb.nElem==result[k].nElem){
    /* Check if all elements are the same (
      Note: both lists are sorted) */
    e1=result[k].elem;
    for (l=0; l<newComb.nElem; l++) {
      if (e[l]!=e1[l]) break;
          /* There is at least one element that is different */
    }
    if (l==newComb.nElem) {
      isRepeated = 1;   break;
    }
  }
}

if (isRepeated!=0){
  #ifdef __SMALLEST_COMBINATION_DEBUG__
  printf(" <= Repeated.\n");
  #endif
  free(newComb.elem);    /* Free memory */
  continue;
}

/* Insert the new combination into the result set */
/* Find the insertion position */
for (pos=0; pos<=lastResult; pos++)
  if (newNum < result[pos].num)
    break;

if (lastResult<lastResultIndex) { /* Append: There is room */
  lastResult++;
  #ifdef __SMALLEST_COMBINATION_DEBUG__
  printf(" <= Append at index %i\n",pos);
  #endif
}
else {  /* Insert: Last result is always gone */
  free(result[lastResult].elem);
```

```
        #ifdef __SMALLEST_COMBINATION_DEBUG__
        printf(" <= Insert at index %i\n",pos);
        #endif
      }

      /* Insert newNum into the result */
      for (k=lastResult; k>pos; k--) result[k]=result[k-1];
      result[pos] = newComb;

      #ifdef __SMALLEST_COMBINATION_DEBUG__
      printf("Index:  Number: Element Index\n");
      for (k=0; k<=lastResult; k++){
        printf("%4d %c%8.2f:",k,(k==pos?'*':' '),result[k].num);
        e = result[k].elem;
        for (l=0; l<result[k].nElem; l++)
          printf(" %3d",e[l]);
        printf("\n");
      }
      printf("\n");
      #endif
    }
  }
}
```