TREBALL DE FI DE CARRERA

TÍTOL DEL TFC: Robust Error Detection for H.264/AVC Using Relation Based Fragile Watermarking

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Telemàtica

AUTOR: Gonzalo Calvar Forte

Supervisora: Olívia Némethová

Director: Prof. Markus Rupp

DATA:

Títol: Robust Error Detection for H.264/AVC Using Relation Based Fragile Watermarking

Autor: Gonzalo Calvar Forte

Supervisora: Olívia Némethová

Director: Prof. Markus Rupp

Data:

Resum

En aquest treball es proposa un nou model de detecció de errors per un stream de vídeo codificat amb H.264/AVC amb una resolució CIF. El model assumeix una modificació dels coeficients obtinguts a cada frame d'un vídeo abans de ser enviat, mitjançant un sistema de comunicacions UMTS.

Al receptor, s'obté un vídeo amb marques, per tal que aquest sigui capaç de sapiguer els coeficients que espera rebre i per tant, poder detectar els errors esdevinguts durant la transmissió, al moment de no rebre el coeficient esperat.

Tots el paquets sense errors poden ser descodificats normalment sense cap problema fins que arriba un paquet amb error, llavors el VLC es desincronitza i per tant cal cridar als mètodes d'ocultació d'errors per tal d'interpolar-los.

Als primers tres capítols es fa una breu introducció als conceptes bàsics del projecte, així com del funcionament del codec H.264, i el principal problema que suposen les paraules VLC., per tal de posar les bases de coneixement per poder entendre correctament aquest treball.

Al apartat 4 es fa referència a diferents tècniques existents capaces de detectar errors.

Al següent apartat s'explica detalladament l'evolució del projecte i els factors mes importants amb els que hi tenim que treballar, desenvolupament de simulacions per obtenir resultats, al igual que l'explicació d'aquests resultats.

Al apartat 6, ens limitem a explicar els canvis i modificacions que hi hem fet al codi H.264.

Finalment, a l'últim capítol es fa una conclusió final, explicant els avantatges que pot generar aquest treball i en quins camps de recerca futurs es podria incloure.

Title: Robust Error Detection for H.264/AVC Using Relation Based Fragile Watermarking

Author: Gonzalo Calvar Forte

Overview

This thesis focuses on the error detection in en entropy encoded H.264/AVC video stream. A bit error within the entropy encoded stream likely causes the desynchronization of the decoder. Therefore, the entire transport layer packet is usually discarded if an error is detected. The loss of a transport layer packet causes considerable visual impairment in the decoded video stream. To prevent discarding the possibly correct parts of such packets, several error resilience methods are known from the literature. One of the possibilities is the resynchronization of the decoder. However, additionally an error detection mechanism is needed to detect the presence of errors in the resynchronized parts of video stream.

In this work, error detection by means of fragile watermarking is investigated. A new fragile watermarking scheme (relation based watermarking) is proposed, evaluated and compared to the state-of-the-art force even watermarking. The performance of both methods is tested with and without entropy code resynchronization for various bit error probabilities. The performance is evaluated by means of rate and distortion. This is necessary since the application of watermarking deteriorates slightly the equality at the encoder and can increase or decrease the compression ratio. The most important is the quality of the video stream at the decoder after being transmitted over an error prone channel with predefined bit error probability.

The results show that the proposed relation based watermarking provides much better quality at the same rate than the state-of-the-art force even watermarking method both with and without resynchronization. This makes the proposed method especially suitable for the transmission over the wireless channels.

INDEX

SECTION 1. INTRODUCTION

H.264/AVC (Advanced Video Coding) is the newest video coding standard, written by the ITU-T Coding Experts Group (VCEG) together with the ISO/IEC Moving Picture Experts Group (MPEG) as the product of a collective partnership effort known as the Joint Video Team (JVT). This standard is especially suitable for low data rate applications. It provides substantial better video quality at those same data rates compared to previous standards (MPEG-2, MPEG-4, H.263) with only a moderate increase of complexity.

These video processing algorithms achieve highly efficient compression by using motion-compensated, DCT-based inter-frame video compression.

The transmission of compressed video streams over wireless communication channels, however, presents several challenging problems that remain to be resolved. One of the most important problems is that on one hand, compressed video streams are extremely sensitive to bit errors, while on the other hand, the time-variant wireless channel introduces randomly burst bit errors due to fading, signal attenuations, and co-channel interference [1], impeding the correct decoding of the received video bitstreams. As a result, the decoder may lose synchronization with the encoder, making it impossible to correctly decode a sequence of Variable Length Coding (VLC) code words until the next resynchronization code word is met.

Predictive coding techniques make the decoding and reconstruction of the video stream at the decoder even more difficult. As the errors in a video frame may propagate to subsequent video frames, the bit errors degrade the quality of not merely individual frames but of the entire video sequence. This work studies methods which improve the error resilience properties of H.264.

The aim of this work is to propose a new error detection scheme. More specifically, the proposed technique belongs to the class of application-oriented error detection schemes, which aim at detecting and locating any errors remaining in the received compressed video stream in order to improve the end-to-end quality.

This work shows the effect of the parity bits used for the error detection with regard to the rate increase and investigates the error detection mechanisms using fragile watermarking. Forced even watermarking (FEW) known from the literature, was originally applied to H.263 discrete cosine transformation (DCT) coefficients. This scheme is adapted in this thesis and implemented for the H.264/AVC Joint Model reference software. To obtain better scalability, an alternative novel relation based watermarking (RBW) scheme is further proposed and compared to FEW. The results clearly confirm its suitability for the error detection purpose. The proposed mechanism considerably outperforms FEW.

SECTION 2. H.264 Codec

On December 2001, ITU-T VCEG and ISO/IEC MPEG [6]-[8] formed *Joint Video Team* (JVT) to finish the H.26L as a jointly project similar to the MPEG-2/H.262.

- For ITU-T, it was given a new standard: Recommendation H.264.
- For MPEG it was named “MPEG-4, part 10”, a separated design from conventional MPEG-4.
- Another name of this compression standard is Advanced Video Coding (AVC) [9]

From that moment H.264 s the newest video coding standard of the ITU-T Coding Experts Group and the ISO/IEC Moving Picture Group. The main task of this standardization effort is to develop a simple and straightforward video coding design. The first goal is offering an enhanced compression performance from its precursors, and the second one is to provide a “network-friendly” video representation which addresses “conversational” (video telephony) and “non-conversational” (storage, broadcast or streaming) applications, normally using in real world.

H.264/AVC [9] has achieved a significant improvement in the rate-distortion efficiency, concerning existing standards using low bandwidth. H.264 offers a number of new features for improved quality and lower bit rates, by means of new techniques, for example: *deblocking filter*, *variable block size motion prediction*, *multiple reference frames*, *weighted prediction*, *switching pictures, reducing the size of the DCT blocks from 8x8 to 4x4* and many others. H.264 handles the variety of applications and underlaying networks using the Video Coding Layer (VCL) which is designed to efficiently represent the video contents and the Network Abstraction Layer (NAL) which formats the VCL representation of the video and provides header information in an appropriate way for conveyance by a variety of transport layer or storage media.

Figure 2.1 shows the structure of H.264 video encoder, which is explained later.

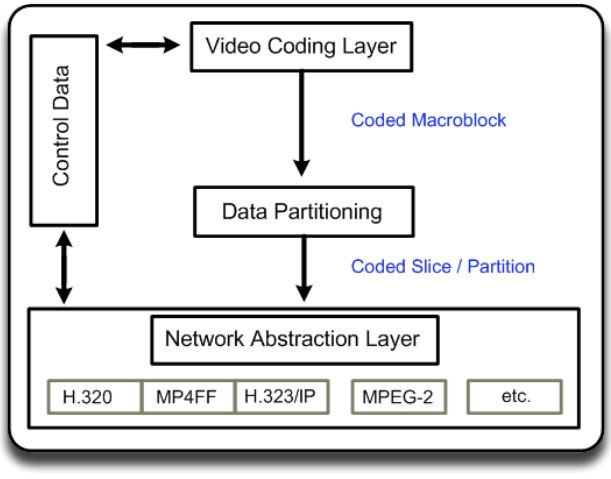


Figure 2.1: *H.264/AVC architecture.*

2.1. Network abstraction layer

NAL is specified to standardize a data format and to provide header information in an appropriate way for conveyance by the transport layers or storage media. All data are contained in NAL units, each of which contains effectively a packet with an integer number of bytes, the first byte belongs to the header. A NAL unit specifies a generic format to be used in both packet-oriented and bitstream systems. The format of NAL units for both packet-oriented transport and bitstream delivery is identical, except that each NAL unit can be preceded by a start code prefix to identify beginning of NAL unit in a bitstream-oriented transport layer.

The NAL facilitates the ability to map H.264/AVC VCL data to transport layers such as:

- RTP/IP: Real-time internet services,
- file formats: e.g., ISO MP4 for storage and MMS,
- H.32X: wired and wireless conversational services,
- MPEG-2: broadcasting.

NAL units are classified into VCL and non-VCL NAL units. VCL NAL units contain the data that represents the values of the samples in the video pictures and the non-VCL NAL units containing any associated additional information such as parameter sets and supplemental enhancement information.
The parameter set contains information which is expected to change rarely and offers the decoding of a large number of VLC NAL units.

2.1.1. Video Coding Layer

The video coding layer of H.264/AVC is similar in spirit to other standards. It consists on a hybrid of temporal and spatial prediction, in conjunction with transform coding. Figure 2.2 shows a block diagram of the video coding layer for a macroblock.

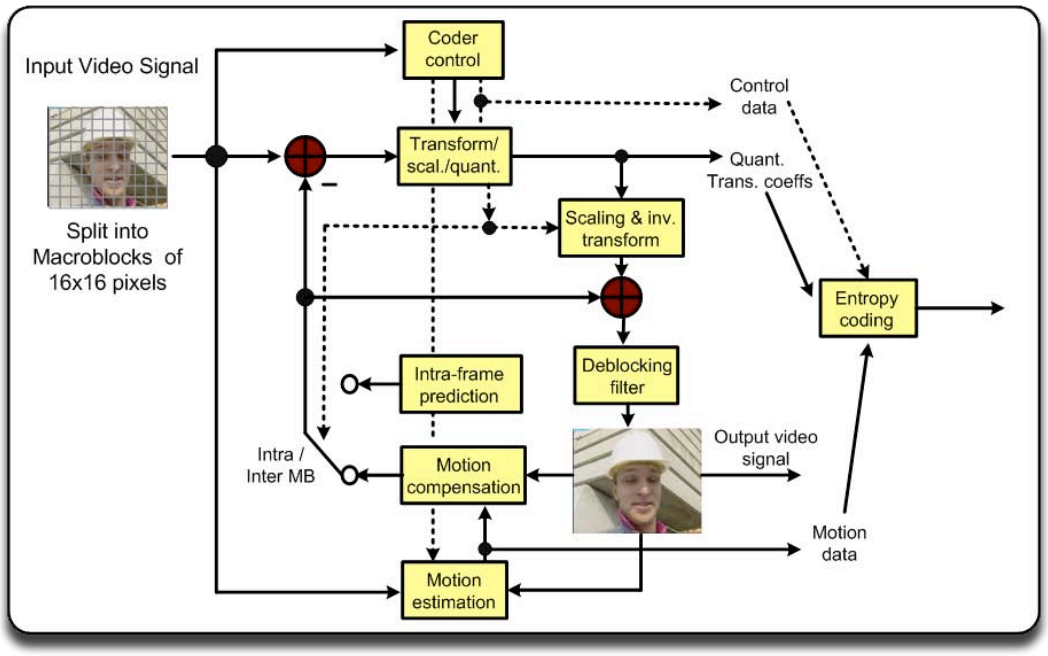


Figure 2.2: *Basic coding structure of H.264/AVC for a macroblock*

In the VCL, each coded picture is represented in block-shaped units of associated luma and chroma samples called **macroblocks (MB)**. There is no single coding element in the VCL that provides the majority of the significant improvement in compression efficiency in relation to prior video coding standards. It is rather a plurality of smaller improvements that add up to the significant gain.

For more details see [11].

2.2. Prediction Modes

The frame is processed in units of a macroblock (corresponding to 16x16 pixels in the original image). Each macroblock is encoded in **intra** or **inter** mode. In either case, a prediction macroblock **B** is formed which is based on a reconstructed frame. In **Intra** mode, **B** is formed by samples in the current frame *n* that have previously been encoded (or decoded and reconstructed at the receiver). In **Inter** mode, **B** is formed by motion-compensated predictions from one or more reference frame(s).

2.2.1. Intra prediction

If a block or macroblock is encoded in intra mode, a prediction block is formed based on previously encoded and reconstructed blocks. This prediction block **B** is subtracted from the current block prior to encoding. For the luminance (luma) samples, **B** may be formed by 4x4 subblocks or by 16x16 macroblocks. There are a total of 9 optional prediction modes for each 4x4 luma block; 4 optional

modes for a 16x16 luma block; and one mode that is always applied to each 4x4 chroma block. Intra predicted macroblocks are sometimes called I-macroblocks (I-MB).

For more information read [12].

2.2.2. Inter prediction

Inter prediction creates a prediction model from one or more previously encoded video frames. The model is formed by shifting samples in the reference frame(s) (motion compensated prediction) to find the best match. It is a temporal prediction and it is the one used for P/B frames. Inter predicted MBs are sometimes called P-macroblocks (if the prediction makes use of the previous frames only) or B-macroblocks (if the prediction was bi-directional).

When MBs are coded as Inter MB, a residual macroblock is obtained from the motion estimation process and the sub-blocks of the residual MB is transformed, quantized and entropy coded.

For more information read [13].

2.3. Transform and Quantization

Each residual macroblock is transformed, quantized and coded. Previous standards such as H.263 used the 8x8 Discrete Cosine Transform (DCT) as the basic transform. The “baseline” profile of H.264 uses three transforms depending on the type of residual data that is to be coded: a transform for the 4x4 array of luma DC coefficients in intra macroblocks (predicted in 16x16 mode), a transform for the 2x2 array of chroma DC coefficients (in any macroblock) and a transform for all other 4x4 blocks in the residual data. With small block size (4x4) the performance loss is reduced.

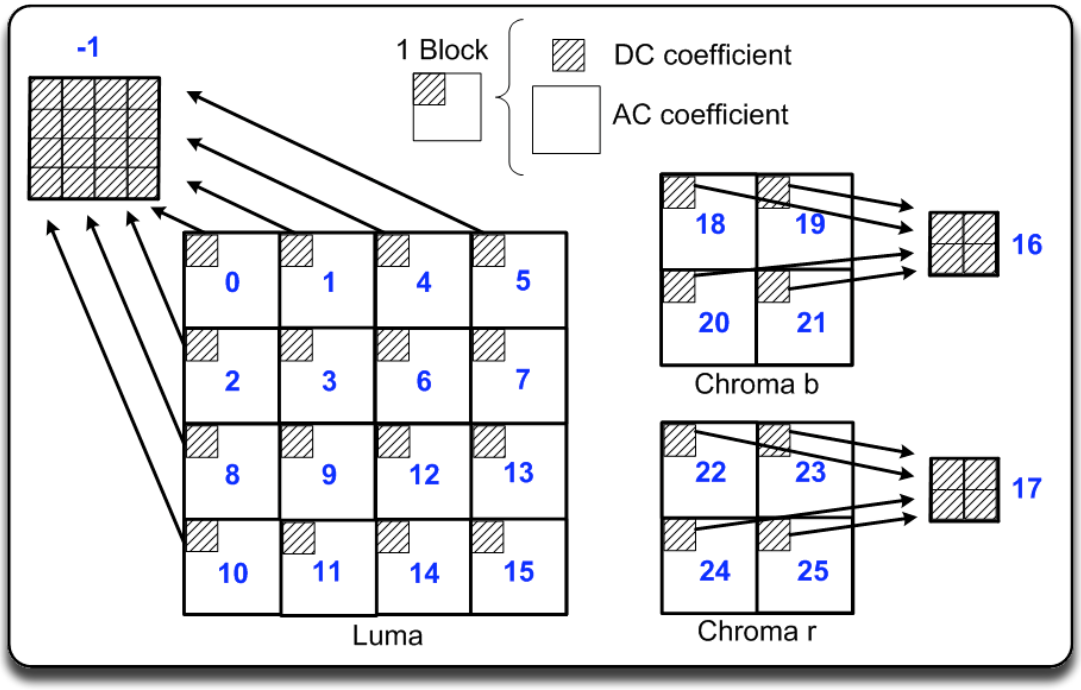


Figure 2.3: *Detail of one Macroblock*

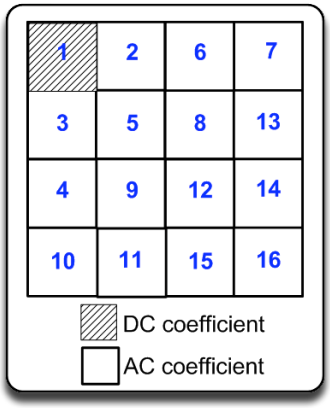


Figure 2.4: *Detail of one luma block*

Data within a macroblock is transmitted as shown in Figure 2.3. If the macroblock is coded in 16x16 intra mode, then the block labelled “-1” is transmitted first, containing the DC coefficient of each 4x4 luma block. Next the luma residual blocks 0-5 are transmitted in the order shown in the Figure 2.4 (with the DC coefficient set to zero in a 16x16 intra macroblock).
Blocks 16 and 17 contain a 2x2 array of DC coefficients from the Cb and Cr chroma components respectively. Finally, chroma residual blocks 18-25 (with zero DC detail of one block coefficients) are sent.

Figures 2.3 and 2.4 about Macroblocks & Blocks also illistrate the zig-zag sequence where is transmitted one entire macroblock, coefficient by coefficient. Zig-zag sequence, can obtain a specific sequential ordering of transform coefficient levels from (approximately) the lowest spatial frequency to the

highest. So, on this way, zig-zag scan assures that the last coefficients of zig-zag will be zero.

2.3.1. Transform

However, in H.264/AVC, the transformation is applied to 4×4 blocks, and instead of a 4×4 discrete cosine transform (DCT), a separable integer transform - with basically the same properties as a 4×4 DCT - is used. Since the inverse transform is defined by exact integer operations, inverse-transform mismatches are avoided. An additional 2×2 transform is applied to the four DC coefficients of each chroma component. If a macroblock is coded in Intra - 16×16 mode, a similar 4×4 transform is performed for the 4×4 DC coefficients of the luma signal. The cascading of block transforms is equivalent to an extension of the transform functions' length.

2.3.2. Quantization

For the quantization of transform coefficients, H.264/AVC uses scalar quantization. One out of 52 quantizers is selected for each macroblock by the Quantization Parameter (QP). The quantizers are arranged so that there is an increase of approximately 12.5% in the quantization step size when incrementing the QP by one. The quantized transform coefficients of a block are generally scanned in a zig-zag fashion and transmitted using entropy coding methods. For blocks that are part of a macroblock coded in field mode, an alternative scanning pattern is used. The 2x2 DC coefficients of the chroma component are scanned in raster-scan order. All transforms in H.264/AVC can be implemented using only additions, and bit-shifting operations, to the 16- bit integer values.

For more information read [14].

2.4. Variable Length Coding (VLC)

There are two methods of entropy coding:

- A low-complexity technique based on the use of CAVLC
- The computationally more demanding algorithm of CABAC

At the slice layer and below, elements are encoded as fixed- or variable-length code - or context-adaptive arithmetic coding (CABAC) depending on the entropy encoding mode. CABAC is not robust enough for mobile networks so this document will be focussed on VLC codes.
When entropy_coding_mode is set to 0, residual block data is coded using a context-adaptive variable length coding (CAVLC) scheme and other variable length coded units are coded using Exp-Golomb codes.

For more information read [15].

2.4.1. Exp-Golomb entropy coding

Exp-Golomb codes (Exponential Golomb codes) are variable length codes with a regular construction.

Each codeword is constructed as follows:

$$[M\ Zeros][1][\text{INFO}]$$

Where INFO is an M-bit field carrying information. The first codeword has no leading zero or trailing INFO; codewords 1 and 2 have a single-bit INFO field; codewords 3-6 have a 2-bit INFO field; and so on. The length of each codeword is (2M+1) bits.

Exp-Golomb is designed to produce short codewords for frequently-ocurring values and longer codewords for less common parameter values.

Table 2.1 lists the first 9 codewords; it is clear from this table that the codewords progress in a logical order.

Table 2.1: *Example of how is assigned a codeword to code_nums*

Code_num	Codeword
0	1
1	010
2	011
3	00100
4	00101
5	00110
6	00111
7	0001000
8	0001001
…	…

2.4.2. Context-based adaptive variable length coding (CAVLC)

This is the method used to encode residual, zig-zag ordered 4x4 (and 2x2) blocks of transform coefficients. CAVLC is designed to take advantage of several characteristics of quantized 4x4 blocks:

1. After prediction, transformation and quantization blocks are typically sparse (containing mostly zeros). CAVLC uses run-level coding to compactly represent strings of zeros.

2. The highest non-zero coefficients after the zigzag scan are often sequences of ±1. CAVLC signals the number of high-frequency ±1 coefficients (“Trailing 1s” or “T1s”) in a compact way.

3. The number of non-zero coefficients in neighbouring blocks is correlated. The number of coefficients is encoded using a look-up table; the choice of look-up table depends on the number of non-zero coefficients in neighbouring blocks.

4. The level of non-zero coefficients tends to be higher at the start of the reordered array (near the DC coefficient) and lower towards the higher frequencies. CAVLC takes advantage of this by adapting the choice of VLC look-up table for the “level” parameter depending on recently-coded level magnitudes.

CAVLC encoding of a block of transform coefficients proceeds as follows.

1. Encode the number of coefficients and trailing ones.
2. Encode the sign of each trailing ±1.
3. Encode the levels of the remaining non-zero coefficients.
4. Encode the total number of zeros before the last coefficient.
5. Encode each run of zeros.

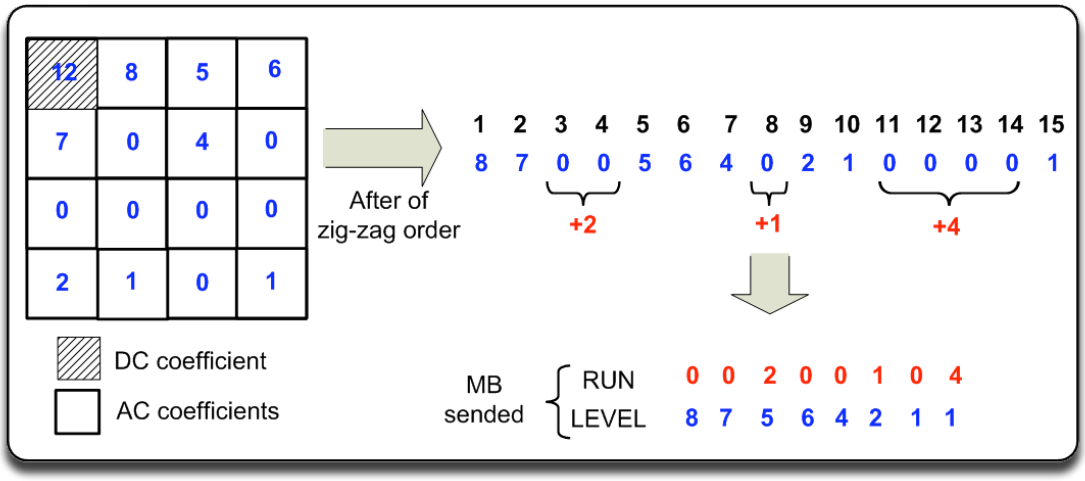


Figure 2.5: *CAVLC squeme*

In order to obtain a lower size of the packet, is necessary to reduce the bitstream size. So, H.264 code uses CALVC mode to fulfil it. As shows Figure 2.5, every MB is scanned following the order of zig-zag (see Figures 2.3 and 2.4), and the system only select the nonzero coefficients, and the method to specify to decoder where is the “zero” coefficients inside one bitstream, is printing for every nonzero coefficient, how many “zeros” are before this coefficient.

SECTION 3. VIDEO IN MOBILE NETWORKS

3.1. Introduction

In recent years, applications of video communications over wireless channels have emerged rapidly. Sophisticated video compression algorithms such as the H.264 ITU [5], and MPEG standards [6]-[8], are used in order to meet the bit rates provided by band-limited wireless channels.

The transmission of compressed video streams over wireless communications channels, however, presents several challenging problems that remain to be resolved.

One of the most important problems is that on one hand, compressed video streams are extremely sensitive to bit errors, while on the other hand, the imperfect wireless channel introduces vast amounts of random and burst bit errors due to fading, shadowing, signal attenuations, and multi-user interference, which results in impeding the correct decoding of the received video bitstreams.

3.2. Error propagation due to use of Variable-Length Code (VLC)

In every video transmission over wireless, the communication between transmitter and receiver uses Variable-Length coded (VLC) bitstreams. VLC are codes having a variable length codeword. They can compact the compressed video bitstream before the transmission. However, VLC codes are very susceptible to errors in the bitstream. As a result, the decoder may lose synchronization with the encoder, making it impossible to correctly decode a sequence of VLC code words until the next resynchronization code word is met. Predictive coding techniques aggravate the situation because decoding errors in a video frame may propagate to subsequent video frames, bit errors degrade the quality of not merely individual frames but of the entire video sequence.

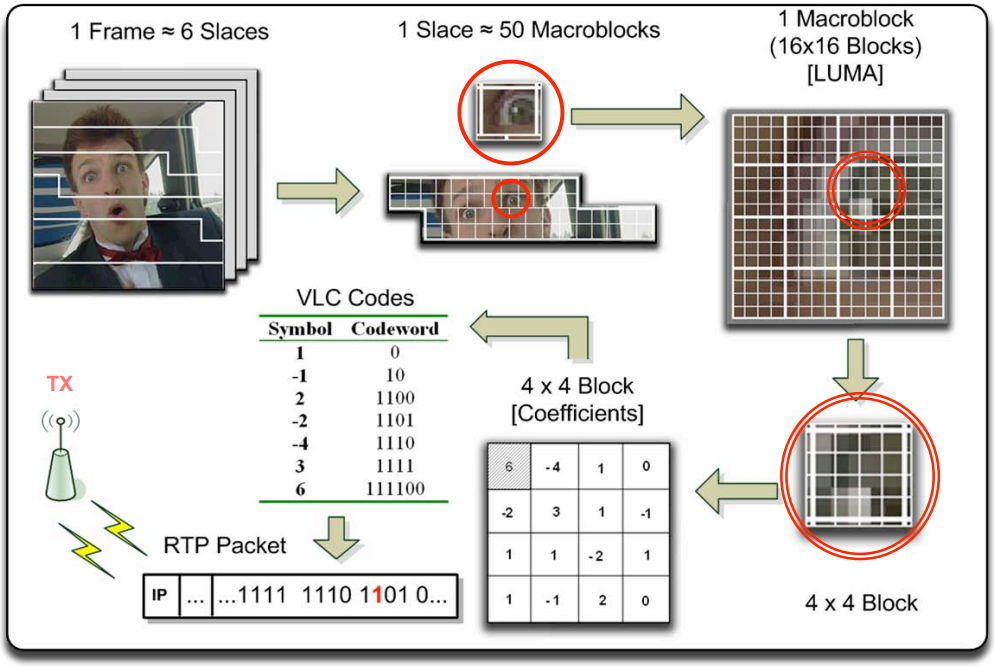


Figure 3.1: *Illustration of some steps that H264 codec follows to send a frame*

As shown in Figure 3.1, the steps to send one video (frame by frame) wirelessly are the following ones:

At first, let us take as example a frame having 6 slices, one slice being composed by 50 MBs (these parameters are the results given by our experiments with QVGA resolution video, but they can be modified, in the configuration of the H.264/AVC encoder).

Then, every MB is composed by 16 blocks. One block (of luminance) has 4 by 4 coefficients. The coefficients are transmitted, following the zig-zag order shown in Figures 2.3 and 2.4. But really, H.264 code doesn't transmit all coefficients inside one Macroblock, so due to prediction mode, it only transmit the difference between the last coefficient into the same frame transmitted and the actual coefficient ready to be sent. In other words, the difference between the actual macroblock and its prediction is then coded, which results in fewer bits to represent the macroblock of interest as compared to when applying the transform directly to the macroblock itself. After the scanning, the coefficients together with another information element are encoded by a VLC. In H.264/AVC baseline profile, a Context Adaptive Variable Length Code (CAVLC) is used, explained previously in 2.4.2.

Next the IP packet with information of the slices is transmitted, over wireless in diagram 3.1's case, so this packet in the Radio Network Control is splitted into several RLC-PDU packets (for the common UMTS networks the size of this RLC packets is 320 bits).

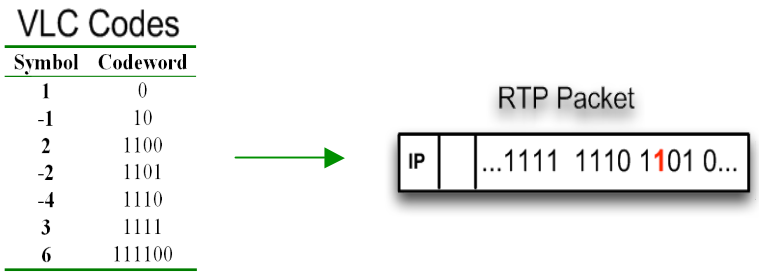


Figure 3.2: *Selecting one symbol per codeword received in a packet*

At this step, it is possible to understand and to observe that VLC codes are very susceptible to bit errors. So when the mobile receives the packets it has to merge the RLC packets in order to get the IP packet; if any error occurs during a transmission, the decoder will be in one of those two situations:

a) Control of errors technique (UDP Checksum), it can detect wrong bits and consequently discard the entire packet. Consequently, the whole slice is lost. There might have been only some single bits wrong but due to VLC, this error possibly propagated over the entire IP packet (size up to 1500 bytes), and not only this, also the error propagated over time until the next synchronization point, which is in this sense the I frame/slice/MB.

b) It is possible to jump Checksum detection technique, however VLC will be desynchronized because the decoder will select an erroneous symbol caused by a codeword (bad codeword) received; as a consequence of a propagated error [16].

H.264 uses temporal references, so if this packet is lost the error will be propagated over time until the next synchronization point.

SECTION 4. PERFORMANCE OF STATE-OF-THE-ART TECHNIQUES

4.1. General approach

In this section different techniques and mechanisms, which try to solve the problems with the discarded packets, erroneous bits, and desynchronization of VLC codewords caused by propagated errors, are explained. All these techniques are based on simple detection mechanism of errors inside of H.264 code.

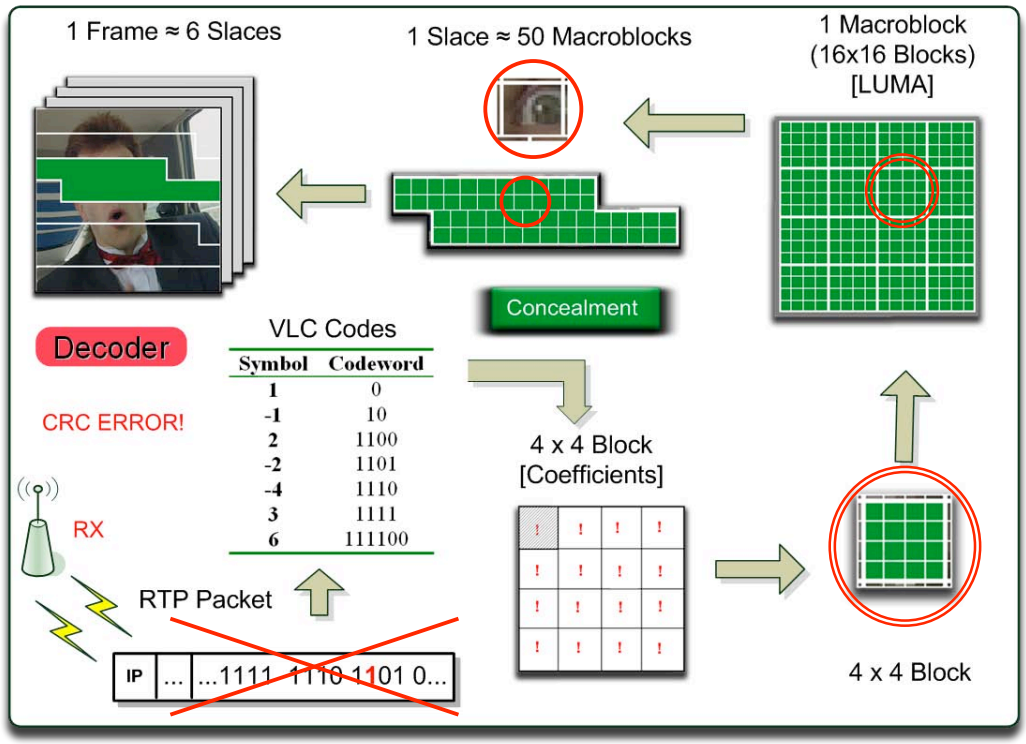


Figure 4.1: *Control of errors problem (CRC)*

In Figure 4.1 the problem of error detection in packet oriented networks is illustrated. When such packet arrives at decoder, if an error occurs during a wireless transmission, (a red 'one' bit in the Figure 4.1), the checksum mechanism of UDP protocol detects the error(s) in the bitstream received, and he discards the whole packet.

As a consequence, the decoder can not know the information of the whole slice (entire packet), and it has to apply an appropriate method of concealment to the whole slice. Such slice can represent an essential part of a picture, especially in the case of low resolution videos, used in wireless communications.

Note that the whole packet is discarded even if possibly only one bit was erroneous. Without any additional complexity or redundancy, it would be

possible to decode the part of a slice until the first error occurrence error-free. However, we do not know the position of the error and the decoding of the desynchronized stream may lead to considerable visual impairments [16].

Therefore, in order to solve this problem, it is extremely important to find the positions of errors inside the bitstream. After that we will be able to use error-free parts of video and to conceal only the erroneous parts.
To facilitate this, it is necessary to create dedicated mechanisms for error detection. The objective of this mechanism is to apply concealment only from the error detected to the end of slice.

As it can be seen in Figure 4.2, such approach is clearly capable of improving the quality. In this case concealment is not applied to the entire slice but only to some MBs.

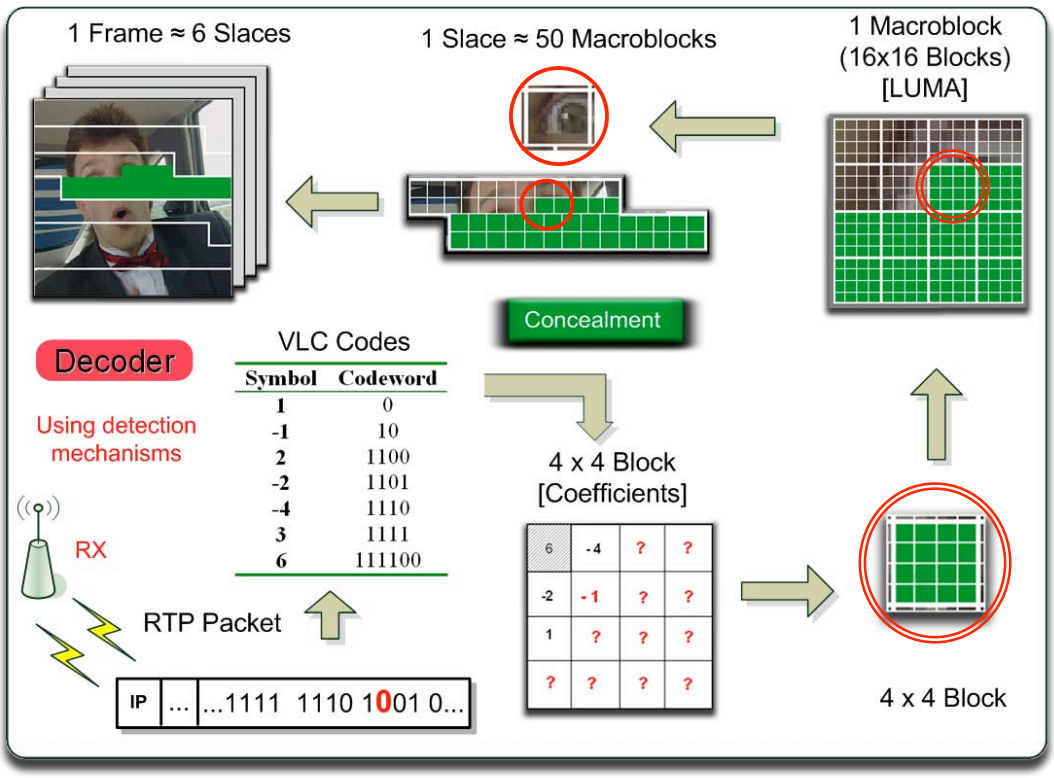
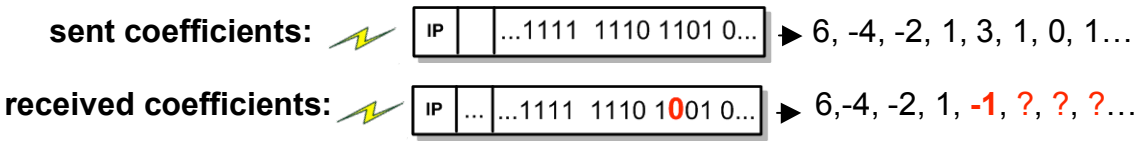


Figure 4.2: *Using detection mechanisms*

In the Figure 4.2 inside the packet, the red 'one' bit belongs to an erroneous bit on a wireless transmission, like in Figure 4.1. In this case, though, the new mechanism *overlook*s the UDP checksum (possible by implementation at the decoder, or explicitly by using UDPLite [17] and it's able to find the error inside the bitstream. Now, as the received coefficients at the decoder are not the expected ones, the mechanism in charge of error detection informs the decoder at the application layer that concealment must be applied from the corrupted MB till the end of the slice.



This way, the decoder can decode the first MB's of the slice, so the first error-free bits of the packet.

At present, several types of detection mechanisms exist to solve the problem of VLC codes caused by propagated errors.

Next subsections explain several error detection schemes. More specifically, these techniques belong to the application-oriented error detection schemes, which aim at detecting and locating any error remaining in the received compressed video stream after FEC (Forward Error Control) decoding. Previous application-oriented error detection schemes essentially detect remaining bit errors by validating the stream syntax and the interpretation given to code words found in the video stream [18].

4.1.1. Error types

Bit errors have different decoding consequences depending on the bitstream components being affected, such as header errors, motion vector errors and quantized DCT coefficient errors or errors in other information elements.
An erroneous header causes subsequent components, including motion vector and quantized DCT coefficient, to be incorrectly decoded. Erroneous motion vectors case motion–compensated information to occur at incorrect spatial positions. Erroneous DCT coefficients cause the luminance and chrominance values of current and following 4 x 4 DCT blocks to be reconstructed erroneously.

Although bitstream errors that effect headers and motion vectors have more severe consequences for the decoding process than errors in quantized DCT coefficients. For that reason we believe that casting additional error detection and localization mechanisms onto quantized DCT coefficients is effective to improve the correct operation of subsequent error concealment.

The packets with detected errors are typically discarded and missing parts of video are subsequently concealed. The reason for this handling is mainly the variable length coding (VLC) [explained in 3.2]. After a bit error, (CA)VLC may easily desynchronize, making the correct distinguishing between the following codewords impossible. Therefore, without any resynchronization mechanism and/or additional detection/decoding mechanisms, the decoding of such may result in considerable visual impairments, or may become even impossible (due to the non-existing code-words, too many or too few left for decoding). The detection of errors allows for utilization of correctly received parts of the packet for the decoding.

4.2. Synchronization Marks

The insertion of synchronization marks is one technique which prevents the error in VLC propagating over the rest of the VLC stream.
So it is possible to add special codewords to the VLC table being these codewords uniquely identifiable within the bitstream. A VLC, a combination of VLCs or any other codeword combination in the bitstream cannot reproduce the synchronization word. Such codewords are usually the longest. However shorter synchronization words introduce less overhead and a trade-off between the robustness and rate has to be found.

Synchronization marks in video stream do not only provide the bitstream synchronization, they can also ensure spatial synchronization at the decoder.

This is achieved by inserting additional fields after the synchronization word for critical information, such as the block address and/or the quantized value.

To limit the errors to a small spatial region, synchronization words may be inserted at various locations, either at a uniform spatial interval in the coded frame or at a non-uniform bit interval in the bitstream.
Using the codewords, if there is an error, this will only propagate until the next synchronization mark.

In H.264 a start code prefix is standardized for the CAVLC code. A unique sequence of three bytes equal to 0x000001 embedded in the byte stream as a prefix to each NAL unit. The location of a start code prefix can be used by a decoder to identify the beginning of a new NAL unit and the end of a previous NAL unit.

4.2.1. Where are the synchronization marks applied?

The standardized start code prefix in H.264 can be used for further improvement of the error resilience and detection. Remember that the synchronization mark is 24 bits (0x000001) long.

The start code prefix only separates the NAL units. One NAL unit usually contains one slice. Still, after the first error in the NAL unit it will not be possible to resynchronize until the next NAL unit is detected. To localize the error more exactly, synchronization marks can also be added in different places within the same NAL unit - slice. There are a lot of possibilities:

- Every RLC PDU (see 3.2) if UMTS is used, or every number of codewords in a stream.
- Every particular number of macroblocks (leads to non uniform placing the marks as different macroblocks have different length after the entropy coding).
- Any other choices.

4.2.1.1. RLC Packet

The Radio Link Control (RLC) layer achieves a very important function over data flow. The RLC protocol provides segmentation and retransmission services for both user and control data. The receiver receives all this packets. It's possible to see that the mobile in his protocol stack has IP upper RLC.

So as soon as the receiver receives the RLC packets from the lower layers it has to merge the RLC packets to obtain an IP packet. A problem occurs if one RLC packet has an invalid CRC such RLC packet will be discarded. When the RLC packet is merged into the IP packet, this causes the UDP checksum calculated over the IP packet fail. The whole IP packet will be discarded, because the position of the error(s) is unknown.
As mentioned in the beginning of this section (see 4.1), H.264 uses temporal references. If an IP packet is lost the error will propagate in time until the next synchronization point.

To improve the performance of H.264 in an error prone environment like mobile networks, passing of the IP packets with detected errors to the higher layers would help essentially.

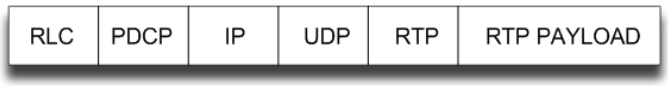


Figure 4.3: RLC packet.

In [19] an improved error detection for H.264 packet video in wireless networks is described, making use of the RLC CRC information. It presents two possibilities: decoding of the correctly received RLC packets up to the first error occurrence, and an additional VLC resynchronization in a combination with improved detection.
Wireless networks also add the CRC to every transport block contained within an RLC packet, it is possible to know which RLC packets are correct if this information passes together with data to the higher layers.

If the content is entropy encoded video stream then, if one RLC packet is lost, all the following RLC packets belonging to the same IP packet will be lost (even if the receiver passes the packets with errors to the decoder). This is a problem that H.264 has because of using CAVLC code. The decoder is not able to resynchronize itself until the next start code prefix.

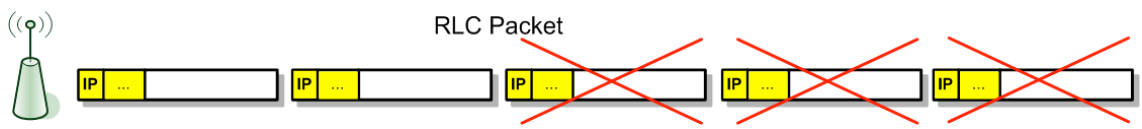


Figure 4.3: *Propagation of error in wireless transmission*

To improve the granularity of the detection it is possible to add a synchronization mark in every RLC packet.
However, this method consists of inserting bits inside of packets so obviously, as consequence, extra overhead will be needed.

So, following the analysis results in [20] after applying synchronization marks on every packet, obtain rate increase between 1 to 10% is obtained depending on the quantization parameter used and the content to be compressed.
A better quality will be obtained after applying this method as a consequence of detecting errors at the decoder, solving the problem of propagated errors as it can be seen in Figure 4.3. One packet is lost, but this one will not affect the following packets. So, the receiver is able to use the correct information and does not have to discard all following RLC packets.

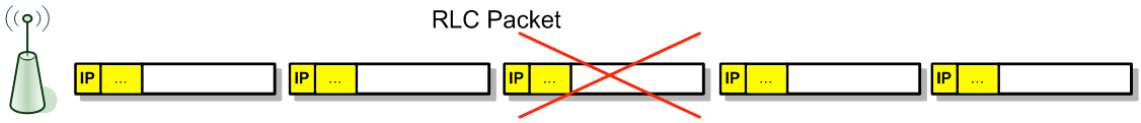


Figure 4.3: *Propagation of error in wireless transmission*

4.2.1.2. Macroblocks

There is another position where synchronization marks can be applied, just in every macroblock, but at expense of having more overhead.

Suppose that one slice of N bytes has frames of 9x11 macroblocks codec. There are 99 macroblocks for each frame; since the slice size is variable the overhead will be studied for different values of it. Normally the slice size will be less than the MTU (Maximum Transmission Unit), the usual MTU (i.e. in Ethernet) is 1500 bytes.

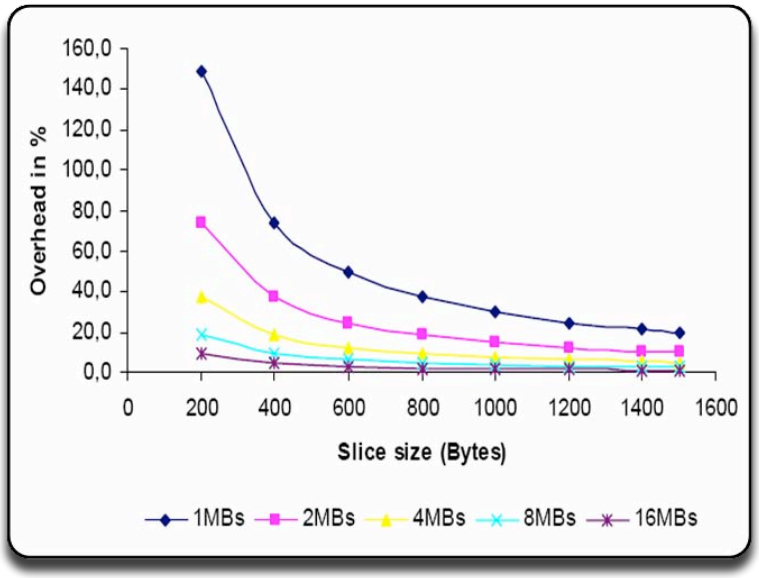


Figure 4.4: *Overhead added for different slice's size and every X MBs.*

The previous figure (from [20]) shows the effect in the overhead added for different slice sizes (adding the synchronization mark every certain number of MBs).

4.2.2. Quality [PSNR] improved after inserting of synchronization marks

In this subsection the importance of error detection within the corrupted packets for improvement of the quality is discussed.

Figure 4.5 illustrates a brief summary of how the quantity of synchronized MBs affects the quality. Let us assume that in this example one frame is composed of 6 slices, so one packet that has the information of one whole slice is 50 MBs long.

- The first image simulates one received frame at the decoder, and the red circles belong to 5 erroneous MB inside the same slice.
- First case (2nd image) simulates one received frame when synchronization marks for every RLC packet are applied (as it is explained in **4.1.1.1**). It can be seen how the first error propagates over the entire packet. Consequently, most part of the packet is lost.
- In next case (3rd image) resynchronization marks are applied every 3 MBs; here it is possible to see that an important improvement of the quality can be obtained. More bits than in case 2 have been added which results in a rate increase.
- The best quality is obtained when inserting resynchronization in every MB, avoiding also propagation of errors as it can be seen in the fourth image (4th). In this case it will be obtained the highest size of the packet.

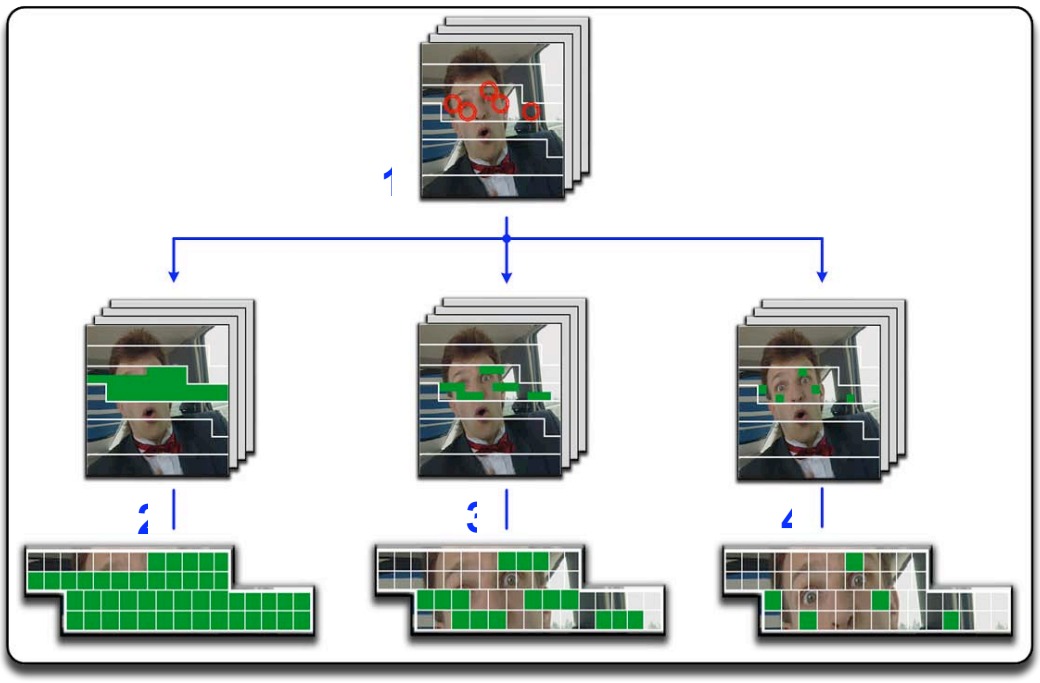


Figure 4.5: *Quality improved depending on MB synchronized*

4.2.2.1. Quality analyzed in H.264

There is a great advantage, regardless of the problems with the overhead, when using synchronization marks as a method for error detection in wireless transmissions.
The advantage is focused in the gain of quality. Obviously, every MB has information of the frame so, the more MB's available to decode, the better quality that will be obtained. For that reason, it is important to detect the errors inside one packet (one slice), in order to discard the erroneous MB and at the same time the information of the non-erroneous MBs inside the corrupted packet.

Next graphs show the results of the analyzed quality in one transmission, after applying synchronization marks at the decoder in one sequence. This analysis has been done over H.264 code with the following parameters in the transmission:

- The sequence (250 frames) belongs to "***CarPhone***" in CIF format.
- The Bit Error Rate (BER) chosen is $1 \cdot 10^{-4}$.
- The quantization parameter QP has been set to two constant values, 16 (the one of the left) and 30 (the one of the right), for both intra- and inter-coded blocks in order to compare the different techniques explained in the following and to avoid the influence of different bit rate control schemes.

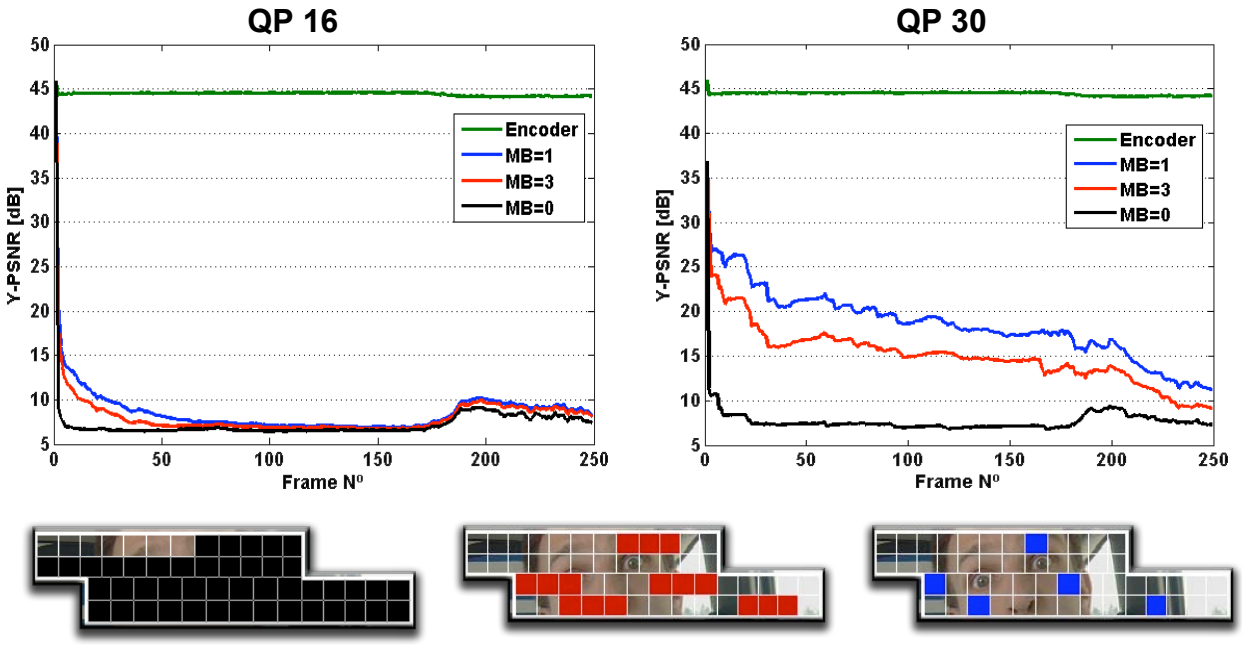


Figure 4.6: *Improvement of quality using synchronization marks at the decoder*

The green line represents the quality at the encoder in both graphs, so there are no errors. That is why it has the highest quality. The Y-PSNR is calculated between the original sequence and its compressed (and possibly degraded) version.

The other lines represent the quality at the decoder after transmission over channel with a Bit Error Rate (BER) of $1 \cdot 10^{-4}$.
The black line has a very low quality because resynchronization marks were not applied and so the errors propagated over the whole slice and subsequently in time.
The red line indicates improvement for the case, where resynchronization marks have been applied every 3 MBs, so the errors only propagate to the next 2 MBs.
The best quality is obtained when applying resynchronization every MB. The blue line shows the result, thus the error propagation of VLC is stopped on the boundary of the following resynchronization mark (next MB). Note, that also overhead needed for the resynchronization grows.

4.3. Parity bits

According to [20], parity bits can be used as a technique for error detection at the application layer. They are inserted inside packets before being sent to the encoder (sender) and after the transmission they are further checked by the receiver to find out if any error occurred. According to the type of code and amount of redundancy it may be possible to localize (and even correct) the errors. The simplest way to detect the odd number of errors is the single parity check code: before sending the bitstream the encoder adds one bit of parity corresponding to the sum modulo 2 (XOR) over the bits in the stream. The decoder can calculate the parity of the received bitstream and compares it with the last bit. If this comparison does not match, the decoder has detected an error. For the example with single parity check code it is obvious that it is not possible to detect even number of errors and that it is not possible to find out how many errors occurred. To do so, more than one bit redundancy would be needed.
There is no standardized method to use parity bits in the bitstream of H.264 to increase the error resilience and enable better localization of the error.

4.3.1. Main functionality

This method is only for detection of errors. The encoder will be able to know where the errors are so it can apply the error concealment later. If an error occurs, it will not be necessary to discard all the information between the synchronization marks because the decoder will be able to know where the error occurred and it does not lose the synchronization.

With VLC codes, if an error happens, the decoder loses the synchronization but if the decoder knows the location of the error, it can discard this erroneous fragment and continue decoding the rest of the transmission, as it was explained in Figure 4.2.

The main idea is to add one parity bit every N bits and checking if these bits had arrived properly at the decoder. With other synchronization marks methods, it is possible to ensure the bits between every 320 bits approximately.

The decoder will receive the bitstream encoded with the VLC code. It will take the bitstream corresponding to the first macroblock, then calculate the parity of this bitstream and compare the calculated parity to the next bits' one, which corresponds to the parity calculated by the encoder. The encoder will be able to detect the error and continue decoding the next macroblock.

4.3.2. Study of overhead

But this method is only efficient when there are only the even errors. If there is an odd number of errors inside a macroblock, the decoder will calculate the parity and will label this as correct. Figure 4.7 shows the results according to [20]. The overhead showed is the result of adding 1 bit for each MB as parity bit control (which can either be odd or even). The graphic compares this overhead to the resulting one when adding more than one bit for each MB.

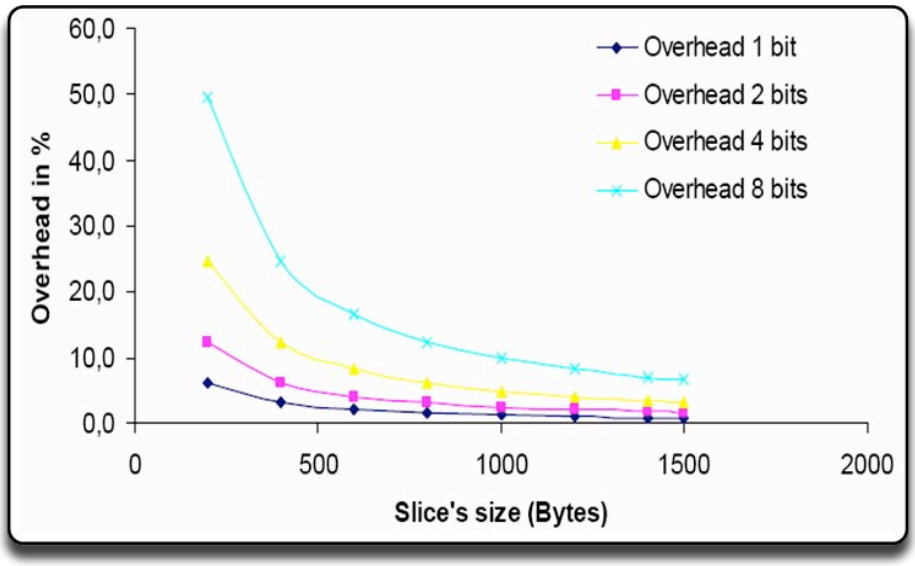


Figure 4.7: *Parity bits overhead for different size of the parity mark inserted every MB.*

4.4. Watermarking

4.4.1. General approach

The arrival of the Internet has resulted in many new opportunities for the creation and delivery of contents in digital form. Applications include electronic advertising, real-time video and audio delivery, Web publishing… An important issue that arises in all these applications is the protection of the rights of all

participants. One such effort that has been attracting increasing interest is based on digital watermarking techniques.
The important issues that arise in the study of watermarking technique are:

- *Capacity:* what is the optimum amount of data that can be embedded in a given signal?
- *Robustness:* How to embed and retrieve data such that it would survive malicious or accidental attempts at removal?
- *Transparency:* How to embed data such that it does not perceptually degrade the underlying content?
- *Security:* How to determine that the embedded information has not been manipulated, forged or even removed?

Nevertheless, watermarking technique should be robust to the following attacks:

- *Common signal processing*: The watermark should still be retrievable even if suffering common signal processing operations, such as resampling, requantization, digital-to-analog and analog-to-digital conversions and common signal enhancements to image contrast or colour.

- *Common geometric distortions:* The watermark should survive geometric operations such as rotation, translation, cropping and scaling.

- *Collusion and forging:* The watermark should be robust to collusion by multiple purchasers who own each one a different watermarked copy of the video sequence. It should also be impossible for colluders to combine their video sequences to generate a different watermark with the intention of framing a third party.

4.4.2. Issues of watermarking

Two categories of watermarking techniques exists named *robust* watermarking and *fragile* watermarking.

Robust watermarking has been applied to embed author and copyright identification in multimedia data [21]. The watermark must be retained in the signal even under intentional signal distortion and intentional attacks to remove it.

In contrast, fragile watermarking refers the process of watermarking a signal in such a way that even the slightest modification of the data causes the extracted watermark to be different from the original. It will be used to add a mark in each slice, MBs or groups of MBs and then, in the encoder, try to find this mark. If the mark is found, the part to be decoded is correct, if it is not found, the part will be labelled as erroneous.

Fragile watermarking can be used to detect tampering of multimedia data.

Although these schemes were designed for still-image authentication, the same idea can be used in the detection of “tampering” of video data by errors in the wireless channel.

4.4.3. Watermarking as a technique of detecting errors

Another technique of error detecting consists on inserting watermark inside the video. Obviously this technique causes a degradation of the quality.

The general idea of the proposed technique is the following one. The video encoder puts a fragile watermarking onto the quantized DCT coefficients before these are passed to the variable length encoder.
The fragile watermarking can be embedded into the data at MB level, forcing the relation between different quantized coefficients in a single DCT block.
At the decoder side, the watermark is detected directly from the quantized DCT coefficients and then, the decoder is able to perform the precise detection and location of the erroneous MBs if it does not receive the expected coefficients.

The structure of the video encoder and decoder using the proposed fragile watermarking technique is shown in Figure 4.8.

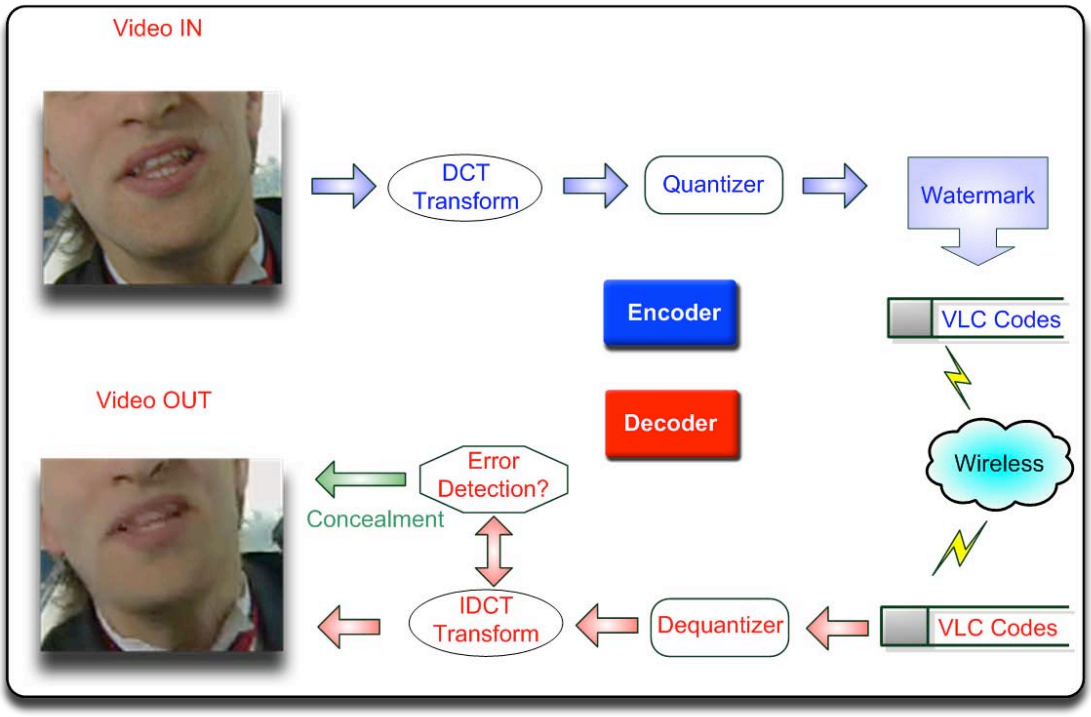


Figure 4.8: *Watermarking technique scheme*

In reality undesired missed detection may happen, so one design goal could be minimizing the probability of the missed detections. By missed detection we understand the case where an error occurred and a detection method was not able to detect it.

The watermark technique here embeds information by modifying quantized DCT coefficients to facilitate the error detection process at the decoder side. Differences introduced by the watermarking of one video frame will show up in the frame differences calculated for the next frame. Therefore, casting a watermark on the DCT coefficients will influence the quality of the encoded video, as it can be seen in Figure 4.8 at the mouth (some quality degradation).

4.4.4. Implemented technique based in fragile watermarking

This subsection explains a fragile watermarking scheme [22] which can greatly improve the error detection capability by using watermark technique. This method is called “Force Even Watermarking” FEW. The scheme is simple to implement and easy to analyze. It is important to know that FEW can just detect the errors, another process will be able to correct them such as concealment techniques but FEW can NOT correct the erroneous MB.

This proposed scheme illustrates the efficiency of error detection by watermarking.
The fragile watermarking scheme proposed in [22] is FEW to indicate that certain DCT coefficients are forced to quantized **even** values.

4.4.4.1. Main functionality

According to [22], the FEW is inserted in each (luminance?) 8×8 pixels H.263 block containing DCT coefficients a_i after quantization. Each a_i with $i = pos$, is transformed in a watermarked coefficient $a_i^{(w)}$according to the following relation, pos (0≤pos≤63) is the cutoff zigzag scan position.

for $i = pos$ **to 63**

$$a_i^{(w)} = \begin{cases} a_i & ; \quad |a_i| \bmod 2 = 0 \ \textit{(is even)} \\ a_i - sign(a_i) & ; \quad |a_i| \bmod 2 = 1 \ \textit{(is odd)} \end{cases} \qquad \textbf{(4.1)}$$

where mod 2 denotes the modulo 2 operation and

$$sign(a_i) = \begin{cases} 1 & ; \quad a_i \geq 0 \\ -1 & ; \quad a_i < 0 \ . \end{cases} \qquad \textbf{(4.2)}$$

In H.264/AVC the DCT transformation is applied to blocks of size 4×4. Then the FEW is applied to these blocks. All quantized DCT coefficients in a 4×4 DCT block after a cut-off zigzag scan position are modified to the nearest smaller even numbers.

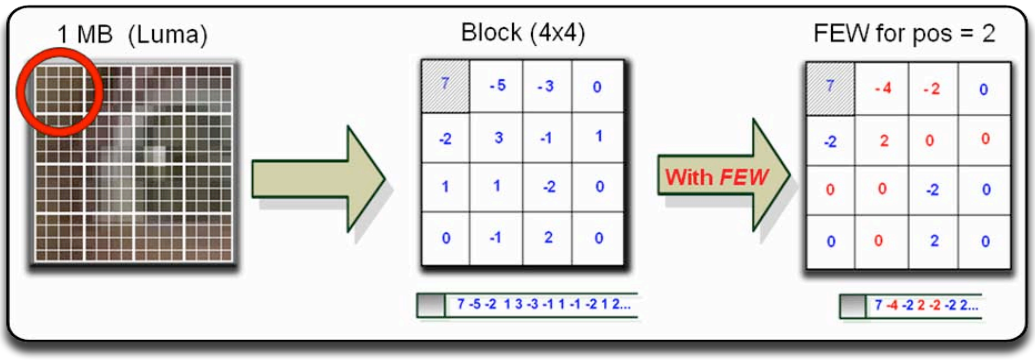


Figure 4.10: *FEW process*

So, in order to properly understand the previous formula, Figure 4.10 represents its general idea.

A brief summary of Figure 4.10 can be a simple scheme where one Luma MB is translated, usually 4x4 blocks, and its coefficients are extracted. After that FEW is applied to all the coefficients and it is possible to see how all odd AC DCT coefficients become even values (the red values are the coefficients which have changed). As it can be seen in the same figure, for DC coefficient number "7" FEW was not applied.
The reason of why the DC coefficient is not modified (first coefficient of the zig-zag process) is because DC coefficients contain the most important information of the MB. So if this coefficient is modified, the quality obtained will be too degraded.

On the other hand, the decoder can detect errors on the way explained hereafter. In the decoding procedure, if a watermarked coefficient is detected as odd, apparently it has been corrupted and therefore the DCT coefficients must have been damaged by channel errors. Hence, an erroneous 4x4 DCT block can be detected and marked for concealment.

Nevertheless, returning to Figure 4.10, the rectangles printed down the two last MBs simulate the information of the packet that will be sent. Then, there are different packet sizes to send the same MB. This is because all the coefficients which value is "1" or "-1", will be Zero ("0") after FEW has been applied to them. And Zeros are encoded in a run length manner [23] and thus no watermarking can be applied in such a case. The errors in the codeword encoding the run can be in most cases successfully detected by a syntax check. So, according to the already explained FEW method, the size of the packet can also be reduced.

4.4.4.2. Probability of error detection

The probability of detection using FEW strongly depends on the starting position *pos* and on the contents of the video sequence since, due to efficient prediction mechanisms, most coefficients (especially the high frequency ones) in the blocks are zero.

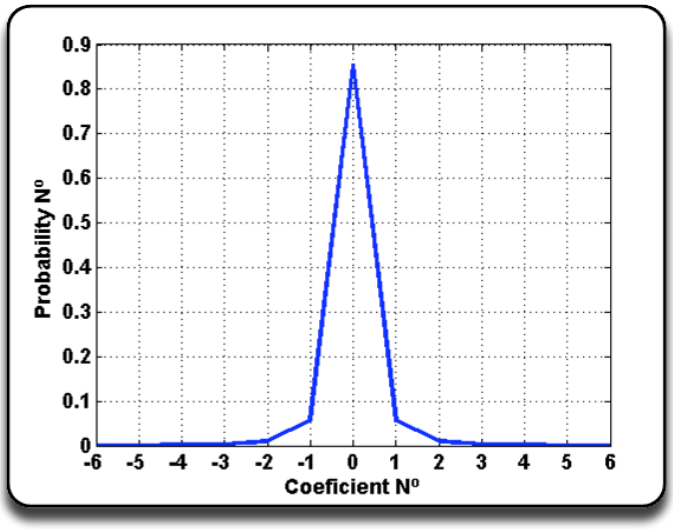


Figure 4.11a: *The probability density function of the coefficient values follow Laplacian distribution, too much values of coefficients are "0" or near to "0". Results are computed using the "CarPhone" sequence; QP=30 (similar graphs were obtained using different QP)*

The probability density function of the coefficient values follows a Laplacian distribution $f(x) = \alpha\, exp(-2\alpha \mid x \mid)$ with parameter α depending on QP [24] (for *QP* = 30 we obtained α = 0.5 for H.264/AVC.

Changing the coefficient numbers in one MB obviously degrades the quality after the video sequence has been reconstructed. The cut-off zig-zag scan position *pos* controls the amount of additional distortion introduced.

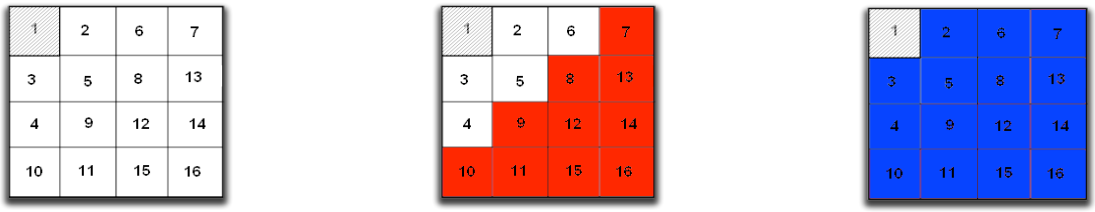


Figure 4.11: *Different start positions in order to begin to apply FEW*

The way to transmit one block is following the zigzag order, like the order of the numbers printed in the MBs in Figure 4.11.

Figure 4.11 shows three cases of *pos* in order to start applying FEW.
Case 1 (White MB): FEW is not applied.
Case 2 (red MB): FEW is applied to half of the MB approximately, beginning in *pos* 7 to *pos* 16.
Case 3 (blue MB): Here FEW is applied to all the coefficients in a MB (all AC coefficients), from *pos* 2 to *pos* 16. Remember that FEW is not applied to DC coefficients.

The two graphs represented below correspond to the “CarPhone” sequence with QP 16 (using *encoder.cfg,* attached in annex A, as configuration settings) after Force Even Watermarking has been applied in two different cases:
Case 1 (left side): Just after FEW has been applied at the encoder, before the sequence has been sent.
Case 2 (right side): a wireless transmission, the sequence has just been received. In this simulation a Bit Error Rate of $1 \cdot 10^{-6}$ was inserted.

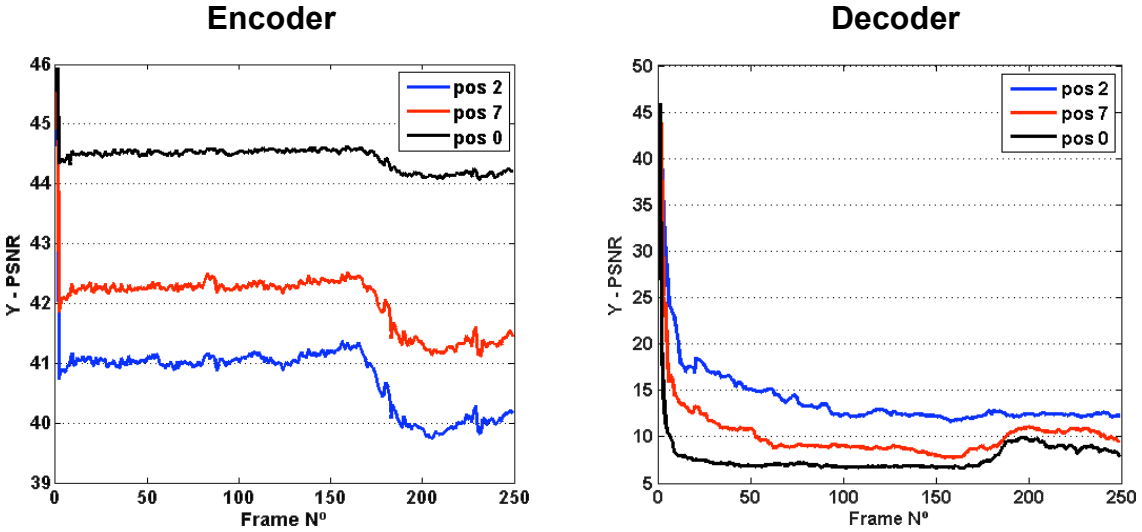


Figure 4.12: *Loss in PSNR after applying FEW Vs the value of pos at the encoder (left) and the decoder (right). “CarPhone” sequence with BER =1·10^{-6} at the decoder.*

Following the same combination of colors as in Figure 4.11, in Figure 4.12, the quality degradation can be seen in the red and blue lines at the encoder. The black lines depicts the case where no FEW was applied. This encoder graph also shows that for a lower *pos* of zigzag (blue line), the greatest degradation will be obtain if no errors occur in a sequence transmission because a lot of coefficients were distributed.

It is obvious that in a wireless transmission several errors may occur. In order to test the robustness and to represent real experiments, video stream transmissions were simulated over an error-prone environment using the Binary Symmetric Channel (BSC) as a model for the wireless communication channel. BSC works as a binary channel of communication where the errors take place of random way. The bit error probability is BEP = {10^{-4}, 10^{-5}, 10^{-6}}. Then, more specifically, in Figure 4.12 the simulation was made with a BER = 10^{-6}.

On the other hand, the pay off of the change at the decoder can clearly be seen in Figure 4.12. In average, an increase of 3.5 dBs PSNR is observed due to the improved error detection capability at the decoder side when decreasing *pos* from 7 to 2.

When FEW is applied to more coefficients, the probability of error detection increases and then a better quality is obtained (blue line) at the decoder sacrificing 3.5 dB PSNR at the encoder. So if there is no FEW and some errors occur during a transmission, they can not be detected and the quality degradation increases (black line).

However there is a trade-off and it is necessary to choose between quality and error detection probability. The more coefficients to which watermarking is applied, the more degradation is obtained but the detection probability increases. And on the opposite, the fewer coefficients with FEW, the less degradation obtained at the encoder, but in this case the error detection probability decreases.

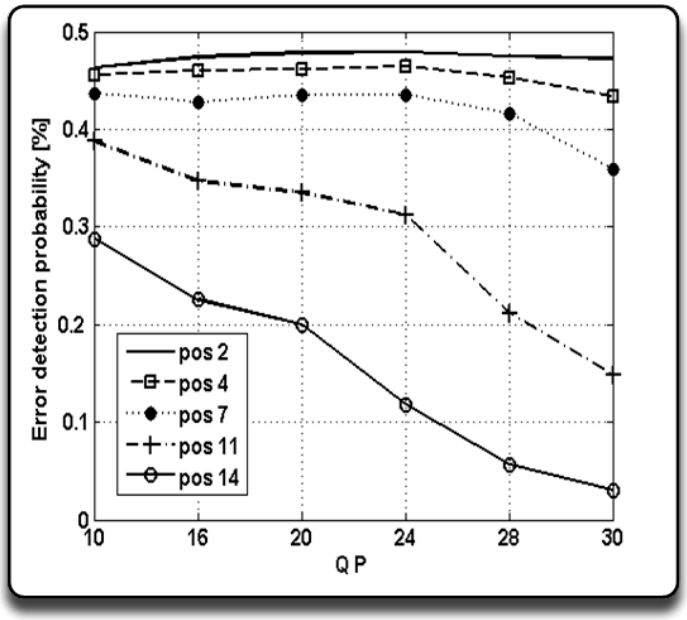


Figure 4.13: *Error detection probability obtained depending on the quantized parameter, for an average FEW applied modifying pos. The results have been computed using the "CarPhone" sequence; channel BER* $1 \cdot 10^{-4}$.

It is also interesting to check the difference in error detection for different values such as *pos* or QP. The "CarPhone" sequence is used in this experiment, too. The results are shown in Figure 4.13. The first observation is that for a fixed QP, the FEW error detection is higher for the low *pos* value, this is because the more watermarked coefficients, the more detection probability obtained. The second notation is that for a fixed *pos*, the FEW error detection capability decreases as the quantization step QP increases. This is because as QP increases, there are more coefficients the indexes of which are larger than *pos* are quantized to zero (0). These quantized coefficients no longer contribute to the fragile watermark. Consequently, when a MB is corrupted, there is a smaller probability for the watermark model to be erroneous, effectively decreasing the error detection capability. An extreme case occurs when QP is bigger.

The way of studying the detection probability in these experiments is easy to implement. Figure 4.14 shows an example of how the "error detection probability" is calculated:

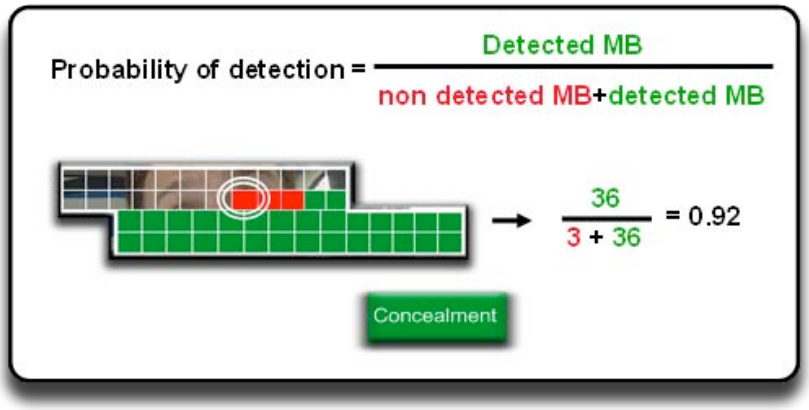


Figure 4.14: *Example of probability of detection procedure*

The procedure becomes clear by looking at Figure 4.14; if any error occurs (white circle), this error may propagate along the packet until the next resynchronization, depending on the error detection method used. So the erroneous MBs (red MBs in the example case) are considered as non detected MBs.
After that, the undetected erroneous MBs and the detected MBs are summed, and these are divided by the detected MB. These detected MBs (green MBs in the figure) are passed to the error concealment routine in order to obtain a better quality.

4.4.5. Conclusions

Using fragile watermarks as a technique to detect errors improves the quality at the decoder considerably. If this method is compared to the two methods explained previously (parity bits and synchronization words), the watermarking technique has the advantage that no overhead is added and only some changes are needed in the transformed coefficients; then, the watermark is detected at the decoder to check that the received watermarked coefficients are the expected ones.
The error detection capability of the decoder can be greatly improved compared to syntax check based error detection schemes. In any corrupted wireless transmission more erroneous DCT blocks can be correctly detected, located and thus concealed. In order to get a better performance in terms of quality, it is necessary to adjust the optimized value of *pos* doing several experiments in the type of the channel and the desired compression ratio.

SECTION 5. PROPOSED METHOD

In [10] it was already shown that FEW means a considerable improvement over the simple syntax check mechanism. However, its detection possibilities are limited. If an error occurs resulting in an even coefficient, it remains undetected; therefore it is not a very robust detection mechanism.
Moreover, FEW allows controlling distortion only by means of changing *pos*. From the point of view of the error detection probability, however, inserting the watermark in the significant regions (low frequencies) is beneficial. To improve the error detection probability, the following section proposes a new scalable fragile watermarking approach, allowing a better distortion control.

5.1. Relation Based Watermarking (RBW)

This subsection proposes a *relation based watermarking* scheme that can greatly improve the error detection capability. The proposed scheme illustrates the efficiency of error detection by watermarking.

The fragile watermarking scheme shown below is RBW, indicating that certain DCT coefficients are forced to modify *n* chosen DCT coefficients $\underline{c} = (c_2, c_3, \ldots, c_n)$ to $\underline{\tilde{c}} = (\tilde{c}_2, \tilde{c}_3, \ldots, \tilde{c}_n)$ in order to fulfil the predefined equation of the extraction function:

$$f(c_1, \tilde{c}_2, \tilde{c}_3, \ldots, \tilde{c}_n) = K. \qquad \textbf{(5.1)}$$

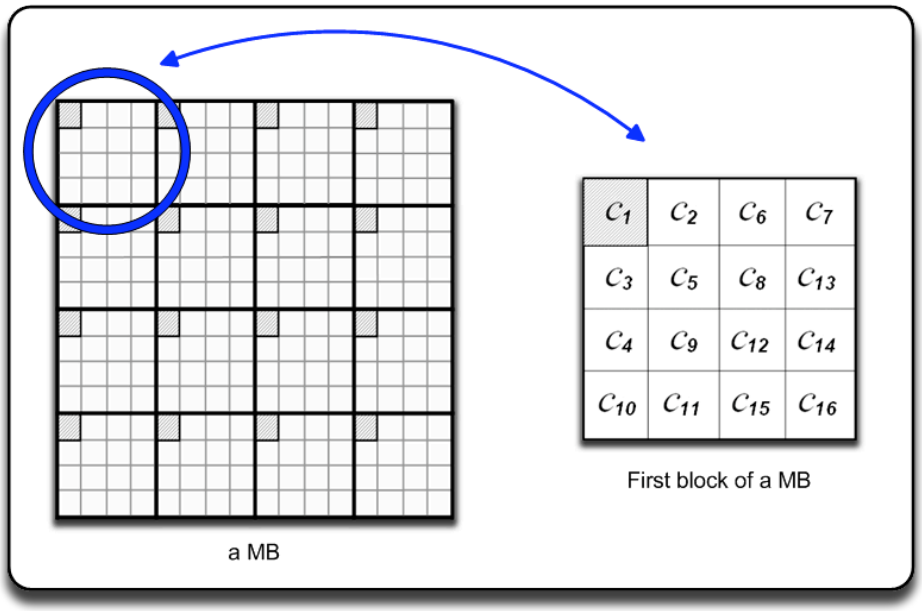


Figure 5.1: *the first block of a MB is selected.*

- Note that c_1 belongs to the DC component used in the extracting function. It remains unchanged to avoid the high distortion its change could cause.
- Coefficients c_2 to c_n are the AC coefficients that can be selected to fulfil the equation.
- Then, the modified coefficients that already fulfil the equation are denoted $\tilde{c}_2$ and $\tilde{c}_n$.
- Variable *n* represents the number of coefficients that will be selected to fulfil the equation.

The value chosen for $\underline{\tilde{c}}$, out of all its possible values, is the one that minimizes distortion

$$\underline{\tilde{c}} = \arg\min_{\underline{\tilde{c}}} \| \underline{\tilde{c}} - \underline{c} \| . \qquad \textbf{(5.2)}$$

The even better can be advantageously replaced by minimization of the distortion in the pixel domain, for example:

$$\underline{\tilde{c}} = \arg\min_{\underline{\tilde{c}}} \| IDCT(\underline{\tilde{c}}) - IDCT(\underline{c}) \| , \qquad \textbf{(5.3)}$$

Where IDCT denotes here the operation of re-scaling and inverse DCT. Quality distortion and detection probability are determined by selecting the extracting function *f* (.).

5.1.1 How does Relation Based Watermarking work?

In order to minimize distortion, a function is required to provide many solutions close to the original coefficients. Thus, the number *n* of the coefficients that can be modified to match the desired extracting function influences the minimum distortion. Fragileness is required for detection probability – functions that change with a single error are necessary as well as a robustness against multiple errors. An *M* modulo operation over the sum of the absolute values of the selected coefficients is chosen in order to meet the criteria:

$$f(c_1, \tilde{c}_2, \tilde{c}_3, \ldots \tilde{c}_n) = \left(|\tilde{c}_1| + \sum_{i=2}^{n} |c_i| \right) \bmod M = K. \qquad \textbf{(5.4)}$$

Increasing *M* improves the fragileness of the scheme against multiple errors but on the other hand it will increase distortion.

It is important to emphasize that all selected coefficients will be used in absolute value.

5.1.2. Selection of parameters

$$\left(\left| c_1 \right| + \sum_{i=2}^{n} \left| \tilde{c}_i \right| \right) \bmod M = K. \qquad \textbf{(5.5)}$$

5.1.2.1. Parameter "n" and "c_x"

This subsection explains the main equation (the equation 5.5) and how the possible parameters (*n, M, k* and c_x) were selected depending on the performance:

- First parameter of the equation (5.5) c_1 is a DC coefficient, so no watermarking is applied to this coefficient.
- Second one, $\tilde{c}_i$, represents all the selected coefficients inside the first block of one MB (see Figure 5.1). The number of coefficients selected is given by the parameter *n.*
  Figure 5.2 shows the different quality degradations when applying RBW at the encoder, selecting more or less coefficients to fulfil the equation.

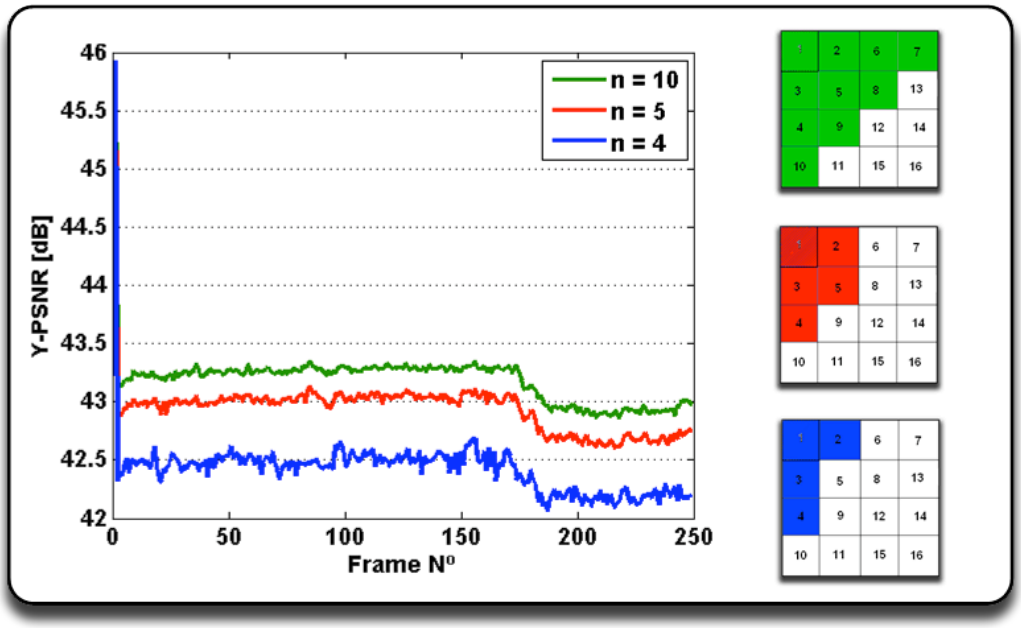


Figure 5.2: *Quality degradation obtained depending on the number of coefficients selected (n) to fulfil the equation, using 16 as a quantization parameter. The results have been computed using the "CarPhone" sequence.*

These results were obtained applying RBW to the "*carphone*" sequence. In Figure 5.2, the Quantization Parameter (QP) was set to 16 because using this

value it is possible to appreciate (separated lines) the difference between all cases. Nevertheless, it has also been analyzed changing the QP value, the results were similar (first *n=10*, second *n=5*, and the last *n=4*).

Obviously these results were expected because, as it was explained in 5.1, the first coefficient of a block belongs to the DC component. So this coefficient and the coefficients near to it following the zigzag order are very important. If they are modified, a much greater distortion would be obtained.
That is why a greater quality degradation is obtained when applying RBW to the blue line. Using less coefficients (first coefficients of the block) to fulfil the equation, as in the blue case, increases the probability of having these first coefficients modified, obtaining consequently a higher distortion.

On the other hand, the more coefficients selected in order to fulfil the equation, like the green line in the previous graph, the higher probability there is of changing the last coefficients of the block. As a consequence, there is lower quality degradation because there is a higher probability of changing the less important coefficients (last coefficients in one block, following the zig-zag order).

Therefore, the simulations in this project were evaluated using the 4 and 6 first coefficients (*n=4* and *n=6)* in order to get the best performance. Nevertheless, the changes will be performed to all the selected AC coefficients so the first coefficient (DC coefficient) is not modified and, in order to diminish distortion, RBW changes the less important coefficients.

5.1.2.2. Parameters "M" and "K"

Now it is important to understand how the parameter *M* is selected because it is the objective of the predefined equation. Figure 5.3 shows the detection probability obtained depending on the value chosen for *M* (the modulo in the equation 5.5):

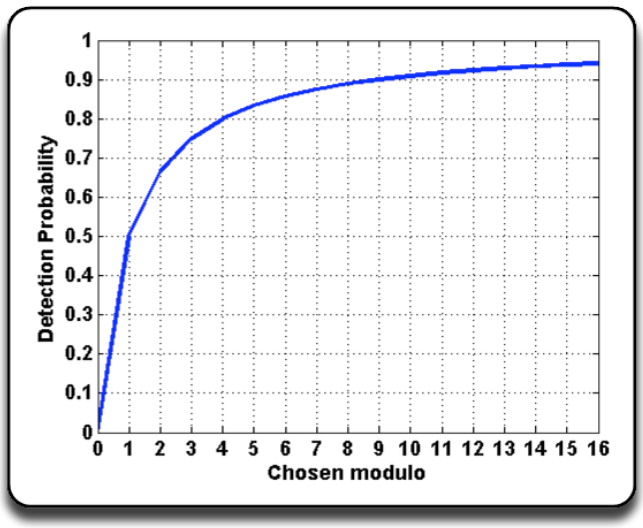


Figure 5.3: *Detection probability obtained depending on the modulo (M) selected to fulfil the equation, and for a fixed number (n) of coefficients selected. The results have been computed using the "CarPhone" sequence;BER = $1 \cdot 10^{-5}$.*

For lower quality degradation the modulo must be low too. On the contrary, to increase the detection probability, *M* should be as high as possible.

So, the best modulo to be chosen is found around numbers 4 or 6, because by using these parameters it is possible to obtain about 80 or 90% of detection probability.

The graph showed in Figure 5.3 is given by the following easy explanation:

If any variable *a* is multiplied by mod *M*, a series of numbers from “0” to “M” {0, 1, 2, 3, ..., *M-1*} will be obtained:

$$a \bmod M = \{0,1,2,3,\ldots,M\} . \quad \textbf{(5.6)}$$

Then, the predefined equation must be equal to *K*, but *K* is a constant so it will be smaller than *M (k*= [*0,M*)) in order to make the equation possible.

Knowing that *K* is a constant number, there is only one possible result for the predefined equation, so the probability of *K* appearing is $\frac{1}{M}$.

Therefore, the detection probability Pd if one error occurs is given by $1-\frac{1}{M}$.

Finally, it is easy to understand that the equation given by graph 5.3 is:

$$\text{Pd} = \frac{M-1}{M} . \quad \textbf{(5.7)}$$

A modulo of 4 and 6 is selected to fulfill the predefined equation.

5.1.2.3. Example of how to find the RBW equation

Hereafter there is a practical example of the RBW with predefined equation (5.5) already explained in Sections 5.1.2.1 and 5.1.2.2:

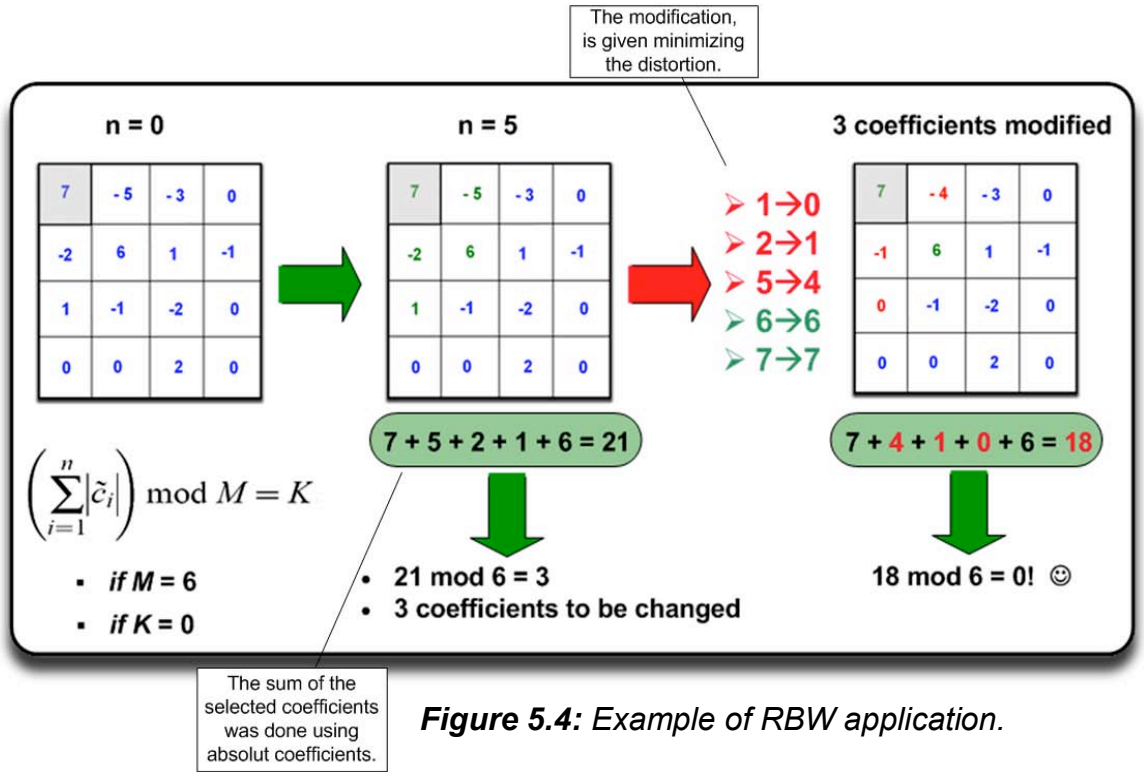


Figure 5.4: *Example of RBW application.*

Following are the steps performed according to Figure 5.4:

- First, the coefficients are selected in the zigzag order (see Figures 2.3 and 2.4), only in the first block of an MB.
- After that, these coefficients are summed together in absolute value.
- Then, modulo *M* is applied to this sum, in this case *M=6*.
    - If the result is equal to *K*, the RBW method does not do anything and continues looking at the next MB.
    - If the result is not equal to *K,* RBW changes the coefficients of this first block of a MB in order to fulfil the predefined equation.
- So, if the result is not equal to *K*, RBW forces the *n* selected numbers, in this case 5, and modifies the coefficients minimizing the distortion. First these coefficients are ordered from smaller to greater. The number of coefficients to be changed is the number obtained in the modulo M sum, in this case 3. The values are changed by adding or subtracting “1”, beginning with the first coefficient (the smaller one) and following the zigzag order. This is an alternative way to solving the optimization problem, that would require higher complexity.
- The number of changes is given by the result of the difference between *K* and the result of the equation after modulo M has been applied.
- Once the changes have been done, the RBW method continues looking at the next MB in order to apply the same routine.

The state diagram below shows the steps followed by the RBW method that were applied in our simulations:

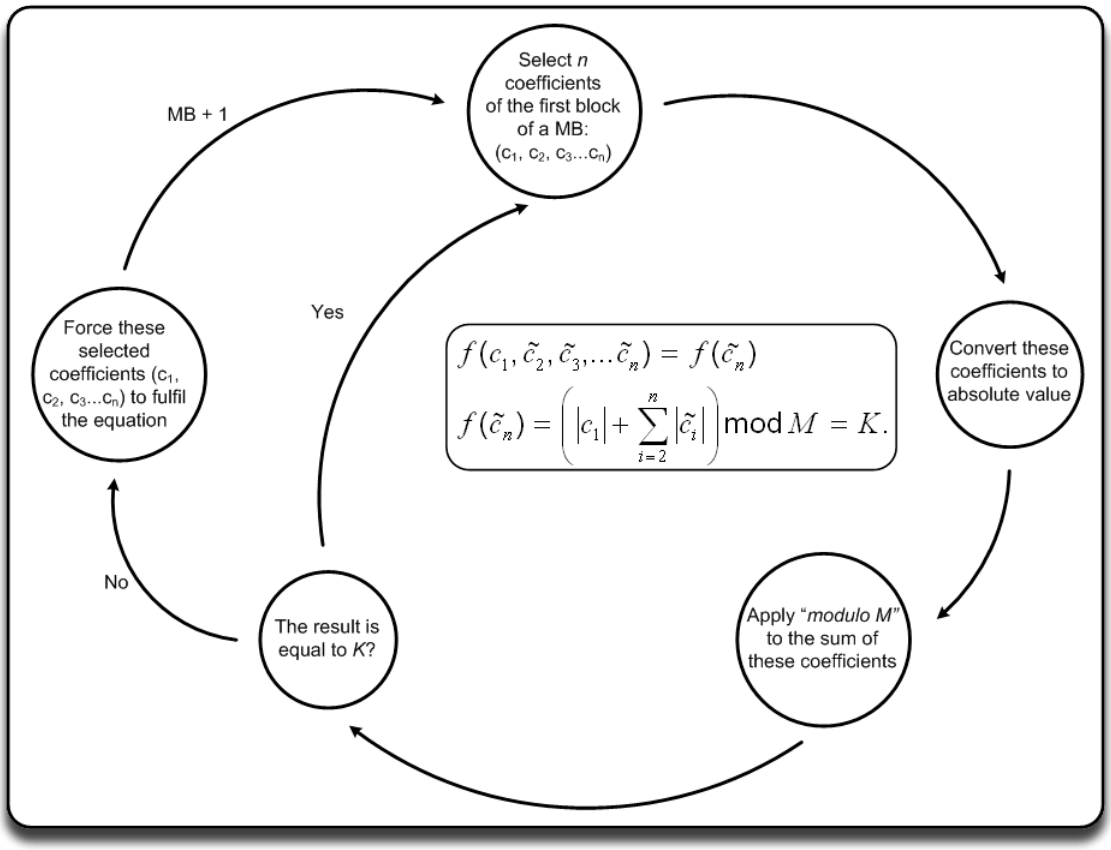


Figure 5.5: *State diagram of the RBW method.*

So Figure 5.5 belongs to only one solution that was used in our simulations but its not shows the higher optimization method. Therefore, in order to obtain the highest optimization it is necessary to make more simulations modifying all parameters to get the less distortion.

5.2. Experiments

A lot of methods and techniques have been studied to improve the error detection in H.264/AVC. In this section a combined method will be proposed. There is not any unique solution to the problem, one possible approach has been proposed here but other combinations are possible.

All simulations were performed applying the configuration of the H.264/AVC code used when testing FEW, so the new method will be evaluated and compared to FEW under the same conditions. A standard video sequence was used in all the simulations. This sequence was *CarPhone* (250 frames). All simulations are in CIF (352x288 pixels) format and have been encoded at a frame rate of 30frames/s. In order to focus on the detection capability of

erroneous quantized DCT coefficients, erroneous bits are only discarded on those coded bits that represent quantized DCT coefficients.

Slicing mode 2 is used with a maximum of 1200 bytes per slice. The sequence has been encoded with different quantization parameters (10, 16, 20, 24, 28, 30) and the rate-distortion optimization for mode selection (implemented by JM encoder) has been disabled. Only the first frame was intra coded (IDR), the rest of the sequence were P frames (B frames were avoided to align with the baseline profile of H.264) in order to show the effect of error propagation. No data partitioning has been used and context adaptive VLC (CAVLC) was chosen as an entropy coding. The Joint Model software [10], with temporal error concealment called copy-paste was used. Copy-paste mechanism conceals the lost part of the frames with the corresponding area of the previous frames. FEW was implemented for every DCT block in a macroblock and RWB only for the first non-zero block in a macroblock. More information about the configuration setup can be found in Annex A.

The error propagation of VLC is stopped on the boundary of slices. As it was explained before, the binary symmetric channel (BSC) was used as a model for the wireless communication channel because, under sufficient error correction and interleaving, any real channel is equivalent to transmitting the unprotected data through a BSC channel. The random bit error of the equivalent BSC channel is in the order of 10^{-4} to get enough strong Forward Error Control (FEC) and enough interleaving. Therefore BER = {10^{-4}, 10^{-5}, 10^{-6}} were used to test the robustness of the error detection using RBW in an error-prone environment.

Moreover, the configuration for the different parameters of the compared methods has been selected to give the highest performance.

The method proposed is a combination of Relation Based Watermarking (RBW) and using VLC resynchronization in order to avoid the error propagation to subsequent macroblocks (MB); however it inserts overhead.

One point that should stay clear about the simulation is that the different sets of *pos (position)* values are not selected optimally for each test sequence respectively. The reason is that choosing different sets of *pos* values for each sequence would result in an overhead to synchronize the encoder and decoder according to the pos values used. So, for each simulation, the *pos* value is changed and it is fixed during the entire simulation, being possible then to compare (in some graphs) how certain *pos* values are better than others. This will make it easy and clear to deploy and understand.

To facilitate the results when showing the improvements of RBW over FEW, several graphs show the improvement between two methods that use the same technique: “Watermarking”.

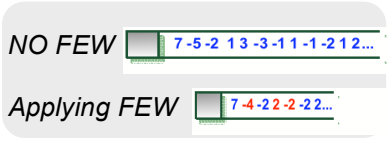
5.2.1. Impact on the rate

Just as it says in Section **4.4.4.1,** after applying FEW to a video stream, the compression becomes more efficient. Mainly, this is due to the high probability of having coefficients with absolute value "one" in the stream (see 4.11a), so after FEW has been applied to these "1" values, they become zero in the watermarked stream.

The reduction $R_{red} = \frac{R - R_{new}}{R} \cdot 100 \left[\%\right]$ of the bitrate *R* (compressed) to a bitrate R_{new} (compressed and watermarked) is shown in Table 5.1.

Table 5.1. *Average bitrate reduction in % after FEW has been applied to the "CarPhone" sequence.*

QP	pos=2	pos=4	pos=7	pos=11	pos=14
10	30.13	27.26	22.08	14.40	7.49
16	42.45	37.32	20.82	16.99	8.01
20	65.92	71.78	20.82	11.40	4.85
24	30.75	24.54	17.17	9.02	3.49
28	31.97	24.10	16.41	7.69	2.73
30	31.48	23.15	15.05	6.94	2.01



Unlike FEW, RBW causes a bitrate increase. The bitrate increase, R_{inc}, is presented in Table 5.2. It is caused by forcing different, and sometimes higher, values to the DCT coefficients.

Table 5.2. *Average bitrate increase in % after RBW has been applied to the "CarPhone" sequence.*

QP	mod = 4			mod = 6		
	n = 4	n = 5	n = 10	n = 4	n = 5	n = 10
10	1.22	2.01	3.09	1.87	1.73	4.38
16	9.30	10.95	5.79	14.44	12.12	17.74
20	25.03	27.85	36.05	44.04	37.70	43.02
24	32.78	34.34	46.57	72.94	63.04	72.07
28	39.66	41.54	55.38	111.62	95.56	107.35
30	47.69	50.23	64.58	151.85	131.25	147.69

5.2.2. Probability of detection

Figures 5.5 and 5.6 show the probability of error detection obtained after using "Force Even Watermarking" (FEW), the existent method. This probability is compared to the results when applying the proposed method, "Relation Based Watermarking" (RBW). The detection probability was calculated following the steps proposed in **4.4.4.2**.

The results show an average for a video sequence in %. Both experimental simulations have been carried out using "CarPhone" as the video sequence. Different QP's (10, 16, 20, 24, 28, 30) are used with the aim of checking the evolution of the performance probability for each QP. The graphs also show the results when applying these techniques and when changing *pos* (in FEW) and *n* (in RBW) variables.

5.2.2.1. Force Even Watermarking scheme

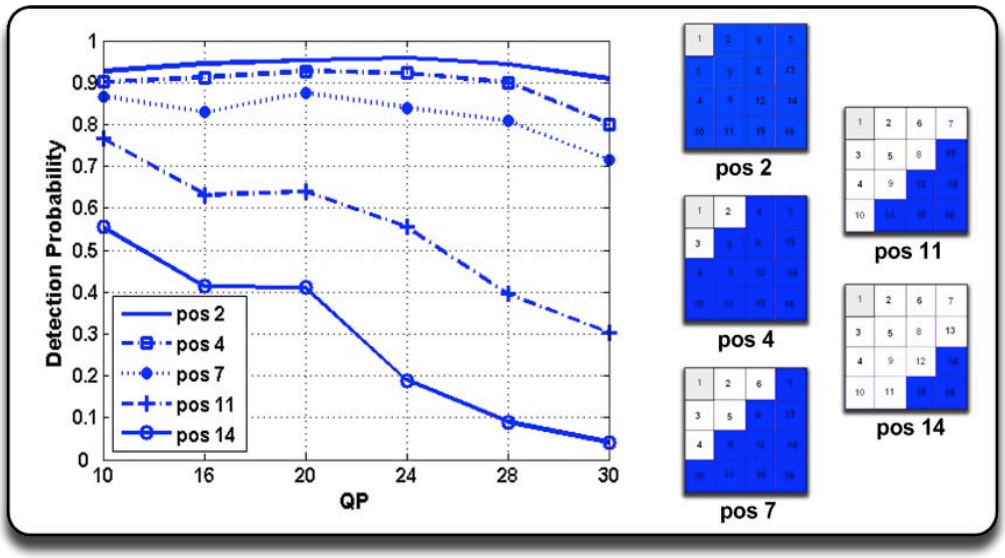


Figure 5.5: *Error detection probability as a function of the quantizer coarseness QP after applying FEW to a video sequence. The results are computed using the "Carphone" sequence; channel BER = 1 · 10^{-5}.*

The main observation in Figure 5.5 is that, if in an erroneous transmission some errors occur (BER = 1·10^{-5}), the more coefficients applied in FEW [pos = 2], the better error detection probability, more than 90%, and as a consequence a better quality is obtained. If FEW is applied to fewer coefficients [pos = 14], the detection probability decreases, from 5% to 55%, and therefore more quality degradation is obtained due to the difficulty in detecting errors, depending on the QP used.

Secondly, for a fixed *pos,* the FEW error detection capability decreases as the quantization step QP increases. This is because as QP increases, more coefficients which values are larger than *pos* are quantized to zero. These quantized coefficients do no longer contribute to the fragile watermarking (see **4.4.4.1**). Consequently, when an MB is corrupted, there is a smaller probability for the watermark pattern to be corrupted, effectively decreasing the error detection capability. An extreme case occurs when QP is very large. All DCT coefficients whose indexes are larger than *pos* are then quantized to 0, and no watermark is embedded at all.

5.2.2.2. Relation Based Watermarking scheme

On the other hand, the probability of error detection is going to be analyzed and compared after applying Relation Based Watermarking (RBW) in a video sequence.

Next graphs show the results of the simulations after applying RBW to the "*CarPhone*" sequence.
The graph on the left is obtained when using modulo 6, while the one on the right was obtained using modulo 4.

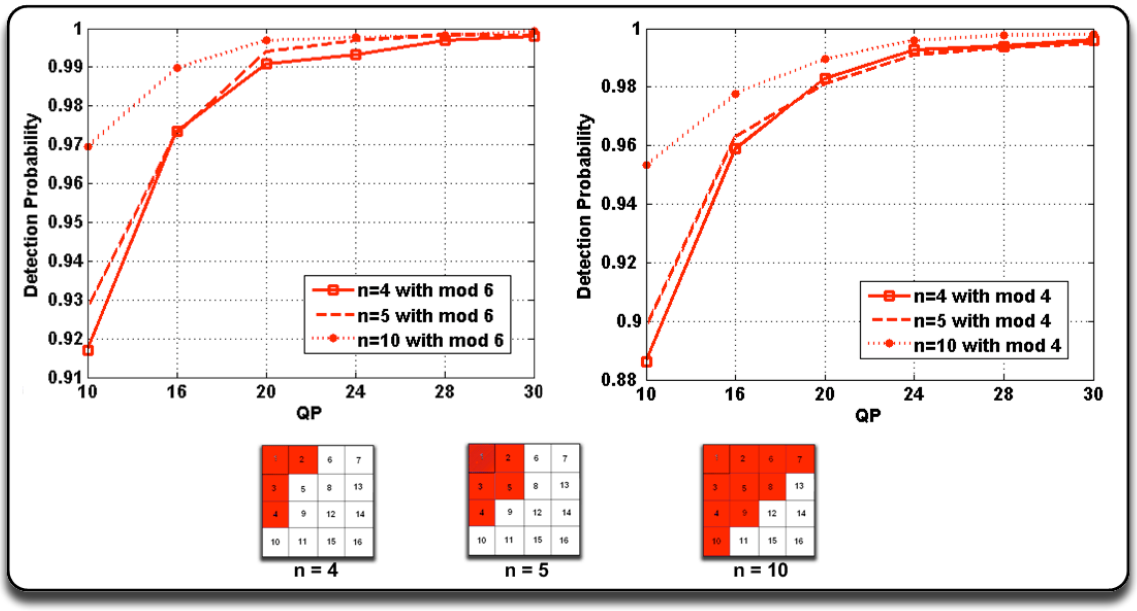


Figure 5.6: *Error detection probability as a function of the quantizer coarseness QP after applying RBW to a video sequence. The results are computed using the "Carphone" sequence; applying modulo 6 on the graph on the left and modulo 4 on the right one; channel BER = 1.10^{5}.*

There is one simple difference between the two graphs shown in Figure 5.6 and it is caused by the modulo used in each one. As it is explained in **5.1.2.2**, the main difference from using a higher or a lower modulo is a notable variation of the quality.

As it can be seen in the left graph in Figure 5.6 (modulo 6), a detection probability from 92% to 97% was obtained depending on the quantity of selected coefficients (variable *n*). So, the results in these simulations coincide with the theoretical explanation in **5.1.2.2**; in the graph on the right, where modulo 4 was used for the simulation, a lower detection probability was obtained, getting a detection probability from 88% to 95%, so using modulo 6 improves it more than a 2%.

In Figure 5.6, there can also be observed the differences obtained when parameter *n* varies. As it is showed in the graph, the more coefficients selected to fulfil the predefined equation (see **5.1.1**), the higher error detection probability is obtained. This is because using fewer coefficients in order to fulfill the equation, it causes a higher distortion probability at the high coefficients (located at the beginning of the zig-zag block) (see **5.1.2.1**).

And finally, for a fixed *n,* the RBW error detection capability increases as the quantization step QP increases.
This is because as QP increases, more coefficients inside a macroblock (MB) are quantized to zero, and according to the explanation in **2.4.2**, the zero coefficients are not transmitted; as a result, these MBs hold a higher error occurring probability.

The opposite situation occurs when QP is low. In one MB too many DCT coefficients are higher than 0, then the most important coefficients (nonzero coefficients) have to be modified and consequently the error detection capability decreases.

5.2.3. Distortion at encoder

To measure the distortion caused by watermarking in the *m*th frame, the luminance peak signal to noise ratio (PSNR) is used as watermarking is only inserted into the luminance:

$$\text{Y-PSNR}[m] = 10 \cdot \log_{10} \frac{(2^q - 1)^2}{\text{MSE}[m]} [\text{dB}] \qquad \textbf{(5.8)}$$

Where *q* represents the number of bits used to express the luminance values, the mean square error (MSE) is given by

$$\text{Y-MSE}[m] = \frac{1}{N_1 \cdot N_2} \sum_{i=1}^{N_1} \sum_{j=1}^{N_1} \left[F_m(i,j) - R_m(i,j) \right]^2 \qquad \textbf{(5.9)}$$

Where $\mathbf{F}_m$ is the degraded frame (compressed and/or watermarked), $\mathbf{R}_m$ is the original frame and N_1 x N_2 is their size.

5.2.3.1. *Force Even Watermarking scheme*

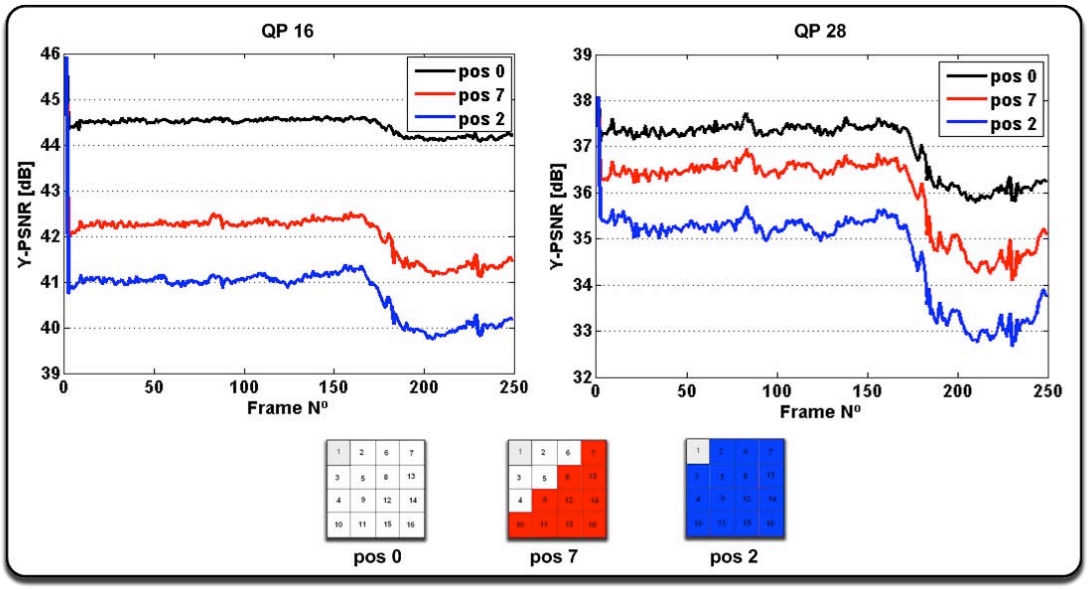


Figure 5.7: *PSNR of the compressed video after using FEW at the encoder. The results of both graphs are computed using the "CarPhone" sequence; using QP 16 on the left, and QP 28 on the right; BER = 1 · 10^{-5}.*

Figure 5.7 shows the encoding PSNR in both graphs. The main difference between them is the way the video sequence video has been encoded: left graph was encoded using 16 as QP parameter, and right graph was encoded using 28 as QP. The difference between those two results is clearly shown in Figure 5.7; for a low QP used for encoding, more bits are used to encode the same video sequence and consequently more quality will be obtained in this video sequence. And vice versa, for a high QP, fewer bits are used to encode the video sequence and therefore less quality will be got in this video sequence.

However, comparing the theoretical explanation to the results obtained in Figure 5.7, they happen to be the expected ones. For a QP of 16, the gain obtained is around 5 or 6 dB greater than when a QP of 30 is used during the whole video sequence.

Another point to be analyzed is the use of different *pos*. The black line shows the case when no watermarking is applied, so this simulation obtained the best quality at the encoder, due to not having forced any coefficient. As it can be seen in Figure 5.7 and according to what was explained in **4.4.4.1**, the more coefficients to which watermarking is applied, the more quality degradation is obtained. So, according to the theory, the blue line, which shows the case when FEW is applied to all the AC coefficients (15 coefficients inside one Block), is 3 dBs smaller (using QP 16) than in the case where no FEW is applied, due to forcing these 15 coefficients to be even values. Note that if FEW is not applied, no distortion is obtained at the encoder and so the best quality is achieved.

Finally, one point that should be clear about the simulation is that only the first frame was intra coded (IDR), the rest of the sequence were P frames in order to show the effect of error propagation. Thus, the high distortion observed in Figure 5.7 between frames 150 and 200 is caused by a big movement within the video sequence, which is due to error propagation (forcing the coefficients to be even values).

5.2.3.2. Relation Based Watermarking scheme

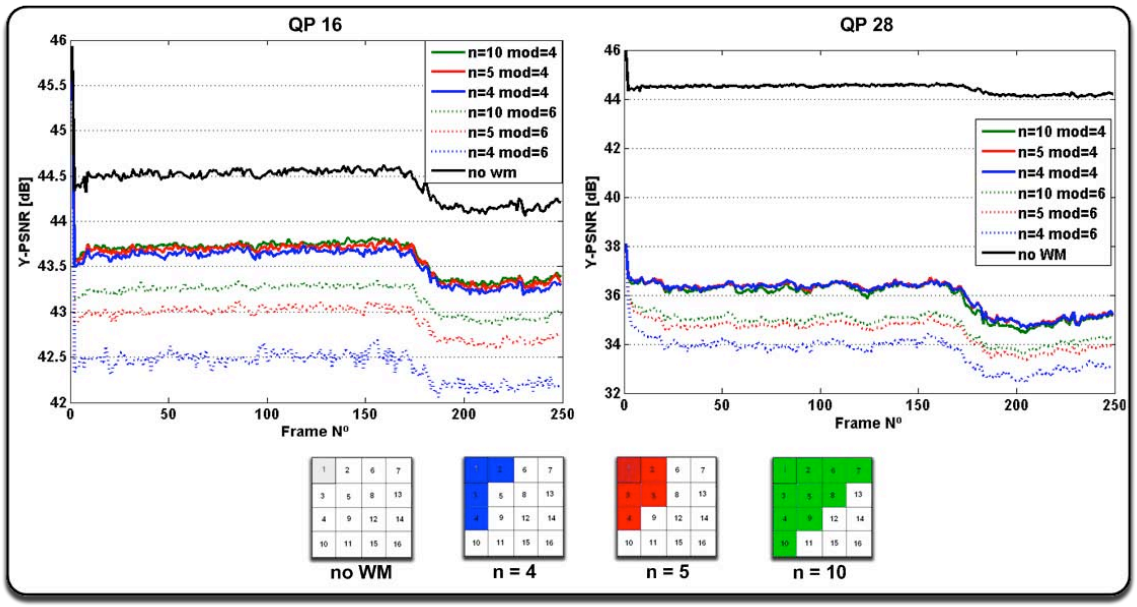


Figure 5.8: *PSNR of the compressed video after using RBW at the encoder. The results of both graphs are computed using the "CarPhone" sequence; using QP 16 on the left graph, and QP 28 on the right; BER = 1 · 10^{-5}.*

The most important difference between the graphs shown in Figure 5.8, is the QP used to encode the same video sequence. The purpose of varying QP is to compare the different results when the new method proposed, RBW, is applied to different compressed qualities. Here, the gain obtained is around 5 or 6 dB higher using a QP of 16 than using a QP of 28, just the same decrement as for the FEW scheme.

Another important difference between the graphs is the difference from applying RBW with modulo 6 (discontinued lines) and with modulo 4 (solid lines).

It can be seen that less quality degradation is obtained when RBW is applied using modulo 4 than using modulo 6. This is because when one value *a* has to be modulo 4 (in order to fulfil the predefined equation), its value can just be 0, 1, 2 or 3 (see Figure 5.9), so the maximum distortion will be 3; on the other hand, if the same value *a* has to be modulo 6, there are more coefficients it can be (0, 1, 2, 3, 4 or 5), therefore, distortion in this case will probably be higher because it can be up to 5 coefficients away.

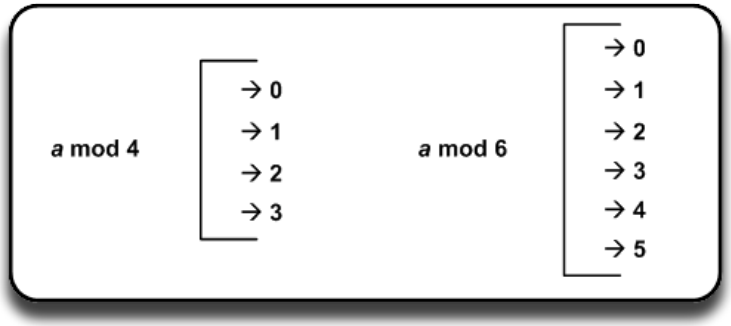


Figure 5.9: *Difference of distortion between modulo 4 or modulo 6 selected.*

Another observation to consider is that for a variable number of coefficients selected *n*, the obtained PSNR is different. According to **5.1.2 and 5.1.2.1**, the first coefficient of one block is the DC component. So this coefficient and the coefficients near it, following the zigzag order, are important enough. If they are modified, the high distortion will be obtained.

That's why higher quality degradation is obtained if RBW is applied to the blue lines (Figure 5.8). Using less coefficients (first coefficients of the block) to fulfil the equation, like in the blue case, the probability for these first coefficients to be modified is higher, consequently obtaining a higher distortion.

The difference in gain between selecting more or less coefficients to fulfil the equation, (10 first coefficients for the green lines or 4 coefficients for the blue lines), is just 1dB.

Note that when RBW is not applied (black lines) no distortion is obtained at the encoder, so the best quality is achieved.

5.2.4. Distortion at the decoder

The most important performance parameter is the quality at the receiver. To test the robustness of the investigated methods, transmissions of the video stream are simulated over an error-prone environment by binary symmetrical channel (BSC) as it is written in 5.2.

5.2.4.1. Force Even Watermarking scheme

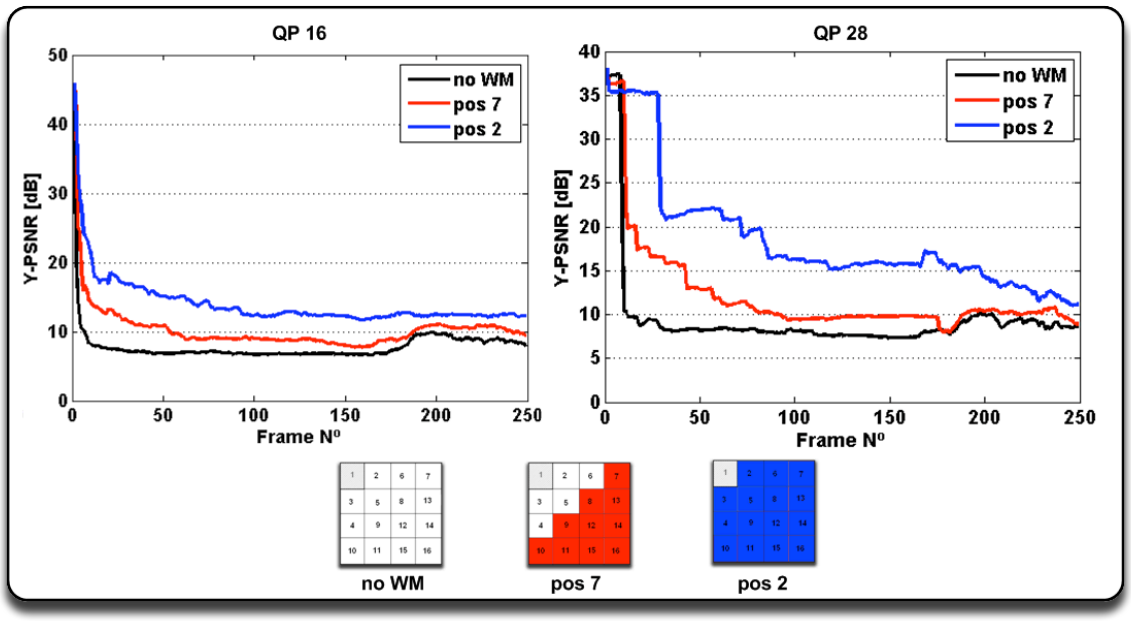


Figure 5.10: *PSNR of the compressed video after using FEW at the decoder. The results of both graphs are computed using the "CarPhone" sequence; using QP 16 on the left graph, and QP 28 at the right one; BER = 1 · 10^{-5}.*

The absolute results for FEW are worse than those presented in [22]. This is caused by the difference between the used codecs and the fact that for the experiments done, the syntax check algorithm has not been implemented yet, what would have improved the performance of FEW and RBW when combined.

Figure 5.10 shows the Y-PSNR with a BER = 10^{-5}, using FEW encoded with QP 16 at the left graph, and QP 28 at the right one. Note that if any error does occur during a transmission, like the results showed in Figure 5.10 for BER of 1.10^{-5}, the video quality received experiences an improvement of around 7 to 8 dB. This improvement depends on the quantity of coefficients where force even watermarking has been applied, on the QP used to encode the sequence and on the quantity of errors occurred. Therefore, it can be seen that the difference between the results at the encoder scheme without BER (Figure 5.7) and at the decoder scheme with BER (Figure 5.10), is that the opposite results occur due to the effectiveness of de FEW method implemented.

It is interesting to see that these results clearly illustrate the tradeoff between the error detection property and quality loss, therefore the more coefficients to which watermarking is applied (the blue lines), the more quality degradation is obtained, but more detection probability is achieved.

And on the opposite, the fewer coefficients to which FEW is applied (red line), the less degradation at the encoder, but less detection probability will be obtained.

5.2.4.2. Relation Based Watermarking scheme

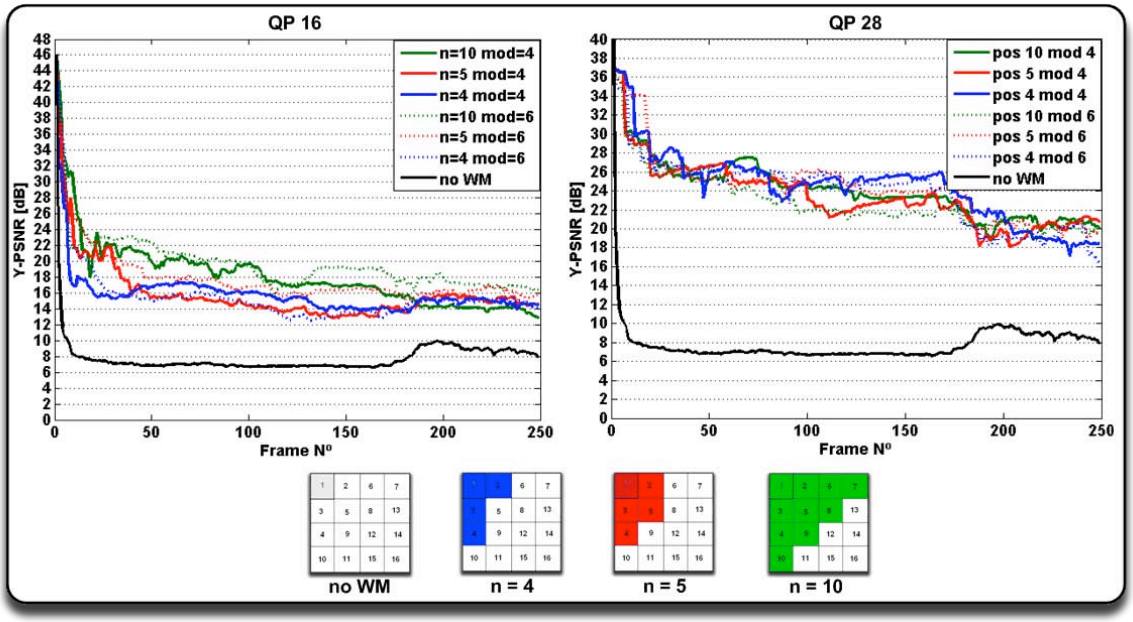


Figure 5.11: *PSNR of the compressed video after using RBW at the decoder. The results of both graphs are computed using the "CarPhone" sequence; using QP 16 on the left graph, and QP 28 on the right graph; BER = $1 \cdot 10^{-5}$.*

The conclusion extracted from these results – under the same error concealment technique –the reconstructed frames from the decoder where RBW scheme has been applied provide better subjective quality due to the significantly stronger error detection ability.

So, the performance differences of the investigated methods can be clearly seen, not only from the decoder quality results, but also from the error detection probabilities.

Therefore, these results obtained with the new method proposed, RBW, show a quality improvement at the decoder of around 6 to 8 dB over the results obtained when using FEW method. Table 5.3 shows a brief summary of the results extracted from Figures 5.10 and 5.11 in order to see how RBW is better than FEW.

Table 5.3. *Gain in dB of Force Even Watermarking method and Relation Based Watermarking, over not applying any detection mechanism in order to detect errors; BER 1 · 10^{-5}.*

QP	***Compared to no detection mechanisms***	***FEW***	RBW
16	Minimum	1 dB	4 dB
16	Average	2 dB	10 dB
16	Maximum	7 dB	16 dB
28	Minimum	1 dB	8 dB
28	Average	5 dB	18 dB
28	Maximum	14 dB	20 dB

5.3. Fairly comparison

To make a fair comparison and to consider the rate increase caused by the use of RBW regarding the rate decrease when using FEW, Figure 5.12 shows how the size of the compressed and watermarked stream depends on the Y-PSNR at the decoder for a BER = 10^{-5}. Still, RBW provides a gain from 2 to 15 dB over FEW's. The gain is slightly higher for the higher error probabilities and would decrease if BER decreased. This gain is given by the higher error detection probability of RBW. For example, when encoding with QP = 20 the error detection probability obtained is 98.2% for a BER = 1·10^{-6}, 98.3% for BER = 1·10^{-5} and 98.6% for BER=1·10^{-4}, all with RBW (Mod=4, and n=4). Using FEW (*pos=4*) the probabilities of error detection were about 90%, similar for all three BER values. Single parity check only results in about 49% of error detections.

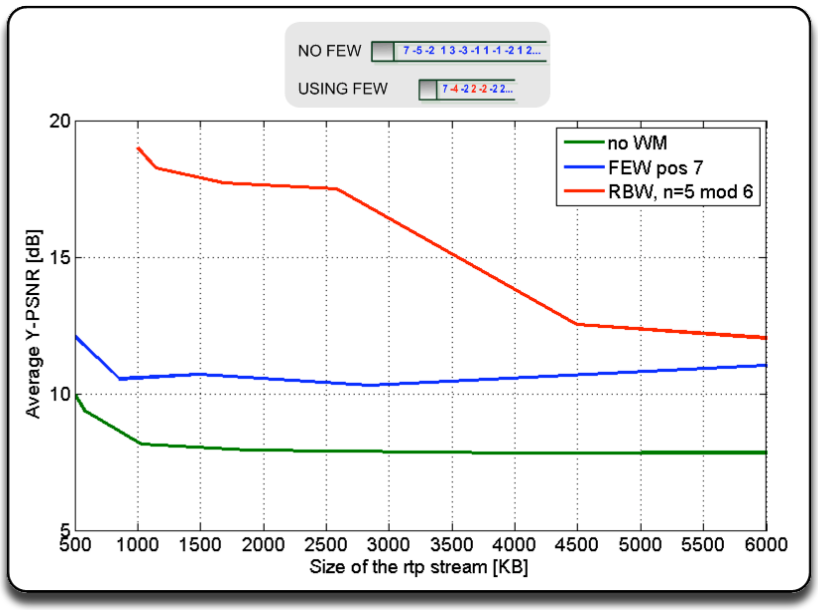


Figure 5.12: *Comparison of FEW, RBW and packetloss concealment: Y-PSNR at the decoder over the size of the encoded and watermarked RTP stream and BER = 1·10^{-5}.*

5.4. Combined method proposed

At the moment, two different detection mechanisms have been presented, both based on the same technique: watermarking. However, these two methods were only created to find the error in order to start applying concealment from the error detection until the end of the slice. Therefore, the error can be propagated until this last MB's (the end of the slice).

Thus, in order to improve the quality, error propagation should be stopped.

One existing method which can solve the error propagation is the inserting of synchronization marks, however this method also increases the rate. The method studied and extracted in [20] was summarized in Section 4.2.

The best simulations with higher quality using the methods analyzed: "FEW and RBW" were selected in order to compare their Y-PSNR to the same simulation but when the method of MB's resynchronization for error propagation avoidance is applied.

Therefore, the variables selected were *pos* = 7 for FEW, *n* = 5 and modulo 6 for RBW, and resynchronize every 1 MB .The best scenario, out of all the existing ones, was selected for this simulation but maintaining the same parameters in order to fairly compare to the simulations made before. Therefore, the sequence has been encoded using a bit error rate of $1 \cdot 10^{-5}$ and a QP of 28.

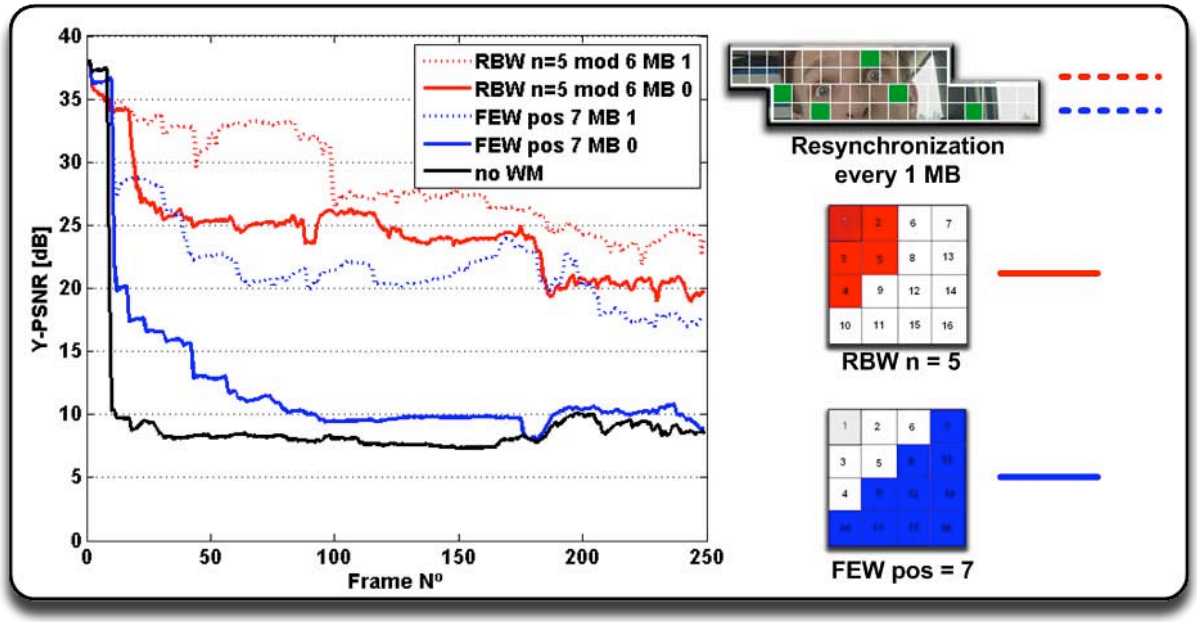


Figure 5.13: *Comparison of Y-PSNR over FEW and RBW using resynchronization marks, for every MB, at the decoder. The "Carphone" sequence was used in this simulation, using a BER = 1·10□5.*

Figure 5.13 shows an important quality improvement when resynchronization marks have been applied to every MB, so avoiding error propagation using either RBW or FEW.

The first observation is that the PSNR of the red lines, which belong to RBW, is about 15 dB higher than the PSNR of the blue lines (FEW).

However, this quality is improved by applying resynchronization marks because the error propagation can be solved. Applying resynchronization every 1 MB to FEW causes a quality improvement of around 10dB over the quality obtained when it is not applied. Applying resynchronization for RBW, the improvement obtained goes from 2 to 8 dBs.

Therefore, if resynchronization marks are applied to every MB at the same time, and the detection mechanism of RBW is applied with *n*=5 and *modulo* 6, it is possible to obtain an important quality improvement at the decoder, more than one average of 21 dB.

So, using resynchronization to avoid propagation does always improve PSNR at the expense of adding some bits.

5.5. Simulations graphics

Finally, quality is not only analyzed by means of the PSNR in the last graphs. This section analyzes some frames belonging to the video simulations done in CIF format in order to compare the quality of the images.
Figure 5.14 shows all the frames extracted from the different simulations, all them were done using the same parameters:
Same Bit Error Rate (BER) = $1 \cdot 10^{-5}$, same video sequence: “CarPhone” in CIF format, same QP = 28, and all the videos were done using intra coded (IDR) frame only at the first frame, the rest of the sequence are P frames. Figure 5.14 also shows the different detection mechanisms in order to “exactly” determine which method is the best one.

Figure 5.14 is structured in two different dimensions:

The timeline of the video sequence is represented downwards, extracting 6 frames:

- Frames 15, 50, 100, 150, 200 and 250.

Different situations (6) are represented across, using the 2 different detection mechanisms combined with another mechanism to avoid error propagation; all cases have been explained along the project.

(A) The first frame of every row shows the video sequence when no detection mechanism has been applied.

(B) Second frame shows the case where Force Even Watermarking (FEW) has been applied from position (*pos)* = 7.

(C) Third frame shows resynchronization marks applied every 1 MB.

(D) Fourth frame was implemented using the new proposed method: Robust Based Watermarking (RBW), using 5 coefficients (n=5) and modulo (mod) = 6 to fulfil the equation.

(E) Fifth frame was composed using a combination of FEW method and resynchronization marks every MB to avoid error propagation.

(F) In last frame a combination of RBW method and resynchronization marks every MB to avoid error propagation was used.

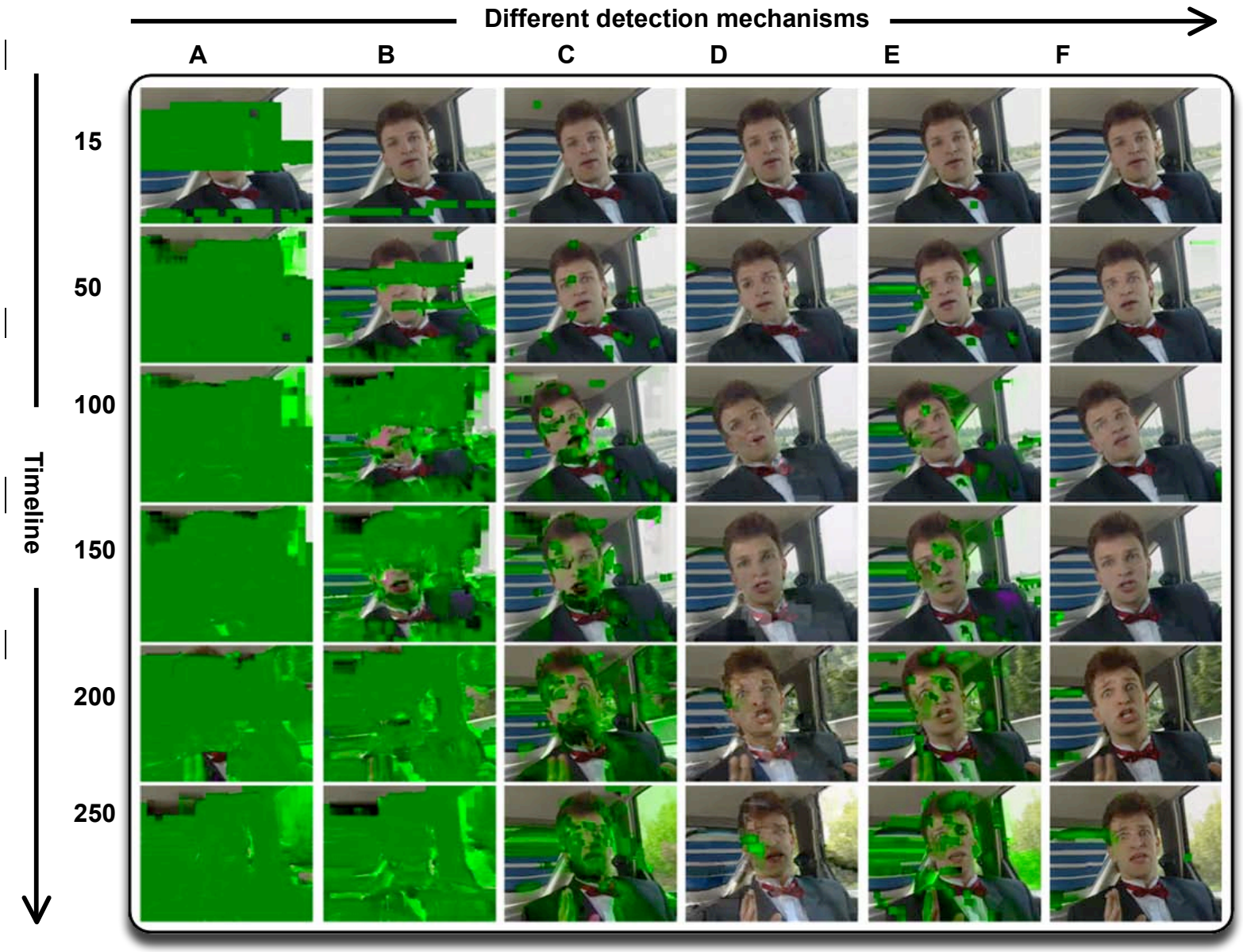


Figure 5.14:
Comparison of the quality improved, at the decoder, using different detection mechanisms with the same characteristics.
Columns show the timeline, and rows show the different techniques implemented.
The sequences are computed using "CarPhone" sequence, Bit Error Rate (BER) = $1 \cdot 10^{-5}$, and QP = 28.

Note that the green MB's shown in Figure 5.14 represent either non-detected erroneous MB's or MB's affected by propagated errors occurred in previous frames. So, when the erroneous MB is detected, the copy & paste technique of concealment is applied.

The main observation extracted from Figure 5.14 is how early (Frame 15) the first video sequence ("A" sequence) is completely damaged, regarding quality, due to not applying any detection mechanism, so the error propagation affects the entire video sequence.

However, after FEW has been applied, the green MBs (non-detected erroneous MB) do not appear until frame 100. Quality has improved due to applying concealment to all detected erroneous MBs ("B" sequence).

In the "C" sequence, resynchronization marks have been applied every MB avoiding so the error propagation. Note that in the first frame, some green MB's do appear alone due to the error occurred just after resynchronization. Step by step, quality falls down because intra coded (IDR) frames were not used along the video sequence. As a consequence, the packet size will increase because of adding some bits [20].

The great results come out for the "D" sequence, where the detection mechanism applied is "RBW". As it can bee seen, just a few green MB's appear along the sequence due to the efficiently of RBW in detecting erroneous MBs. On the other hand, last frames (Frames 200-250) suffer some quality degradation due to the copy of the previous frame given by concealment method. However, quality will be better in an error-free transmission, as it can bee seen in Figure 5.8 (PSNR).

At "E" and "F" simulations, the recommended combined method has been applied, which tries to improve quality at the decoder as it is explained in 4.2.

Therefore, using FEW as the detection mechanism ("E" sequence) together with resynchronization every MB, the obtained quality is better than if FEW is applied alone, without solving the error propagation (sequence "B"). It may also improve quality comparing to apply resynchronization marks alone, without applying detection mechanisms as shown in sequence "C".

And finally, the best quality is achieved in sequence "F", where resynchronization marks were applied, in order to avoid error propagation, together with a sequence which is using RBW to detect the errors.

RBW provides more than 90% of error detection, and being able to resynchronize the sequence, avoids the error propagation whenever an erroneous MB is not detected. Note how scarcely green MB's appear.

It has been clearly demonstrated that RBW, the new method proposed for error detection, is better over all the other existing detection mechanisms, and it can be improved by applying resynchronization marks in order to avoid the error propagation.

SECTION 6. CHANGES TO THE JOINT MODEL CODE.

For the simulations done, the Joint Model H.264 [25] version 10.1 has been used. This software is freely available to the user without any license fee or royalty. Generated by the JVT this software is composed by a H.264/AVC video encoder and decoder, and all the source code is included in the package.

The encoder has been modified by applying the watermarking techniques, so it has been used to generate the RTP video streaming. The encoder has been adapted just as the proposed method explained in 5.1 to obtain the video stream (Table 6.1 shows the main parameters set at the configuration file).

```
FrameRate       =  30.0  # Frame Rate per second (0.1-100.0)
SourceWidth     =  352   # Frame width
SourceHeight    =  288   # Frame height
IntraPeriod     =  0     # Period of I-Frames (0=only first)
NumberBFrames   =  0     # Number of B coded frames inserted (0=not used)
SymbolMode      =  0     # Symbol mode (Entropy coding method: 0=UVLC, 1=CABAC)
OutFileMode     =  1     # Output file mode, 0:Annex B, 1:RTP
PartitionMode   =  0     # Partition Mode, 0: no DP, 1: 3 Partitions per Slice
SliceMode       =  2     # Slice mode (0=off 1=fixed #mb in slice 2=fixed #bytes in slice 3=use callback)
SliceArgument   =  1200  # Slice argument (Arguments to modes 1 and 2 above)
```

Table 6.1: Important values changed in the configuration file of the encoder.

The decoder has another configuration file, but it is less complex than the encoder's case. It only needs to know which video stream is going to be decoded, which concealment method is going to be used and set the NAL mode to RTP packets. Both configuration files, encoder and decoder, are shown in the appendix.

When all the parameters are introduced in the configuration file, the encoder software is executed to obtain the RTP stream of the video with the specified characteristics.

That stream will be the input of the modified decoder software. Modified because new characteristics have been introduced to the source code of that decoder in order to obtain the required outputs to achieve the purpose, i.e. of inserting the watermarking by modifying some coefficients, to insert the supposed errors into MB's in the stream, and to obtain the information about the stream structure necessary for the predictor design as will be shown later.

Each syntax structure in H.264/AVC is placed into a logical data packet called a NAL unit. All NAL units following the first I frame (IDR) have a slice or a type of data partitioning. In the simulations done here, no usage of data partitioning is assumed, thus every NAL unit represent a whole slice, and every slice is one part of the bitstream in the Joint Model, with VLC synchronized at the beginning.

6.1. Introduction

For the simulations it is necessary to modify all MB's located in "*predicted Frames*", therefore, the first frame of the sequences (Frame 0) belongs to IDR frame and so it will not be modified.

Other important information to be considered is that watermarking will only be inserted to Luma coefficients; Chroma coefficients will not be modified.

Every sequence, the H.264 code completes the same continuous cycle for each slice, and so for each packet.

The cycle begins in Frame 1 (because frame 0 is not modified), and at the first MB (MB 0). Inside this cycle, the MB's are codified one to one until the last MB of the slice.

The number of MB's inside one slice depends on the QP and depends on the maximum size of the slice, this value is represented in Table 6.1: Slice size = 1200 bytes. Therefore, every slice (every packet) is 1200 bytes long. As QP decreases, quality increases, therefore more MB's are obtained and so, less MB's there are in a slice. When the cycle is finished, a new RTP stream is built. The H.264 code begins a new cycle with the next slice.

In the simulation, every frame has 396 MB's, this value is given by the type of the video codification: CIF format. When the first 396 MB's are codified, the H.264 code continues codifying the next frame. So, the encoder process is finished when all frames of one sequence are processed. The number of frames to be encoded is part of the information included in encoder.cfg. In the simulations, this value is fixed at 250 (0-249), as it can bee seen in table 6.1.

6.2. The encoder

6.2.1. Store the quantified coefficients

As Figure 4.8 shows, watermarking is inserted at the encoder, but only into the quantized coefficients of the luminance component so it is necessary to find them in order to apply watermarking. These coefficients are located at function "DCT_LUMA" of *block.c* file.

Within every cycle, MB's are encoded in the scan order. The coefficients of every MB are stored just after the quantifying step.

In the following a "*for*" cycle is extracted from the JM DCT_LUMA function in *block.c* file.

As it can be seen, this *for* runs all quantized coefficients, which are stored at the end of the *for*, in the variable *c16*:

```
for (coeff_ctr=0;coeff_ctr < 16;coeff_ctr++)
  {
    if (is_field_mode)
    {
      // Alternate scan for field coding
      i=FIELD_SCAN[coeff_ctr][0];
      j=FIELD_SCAN[coeff_ctr][1];
    }
    else
    {
      i=SNGL_SCAN[coeff_ctr][0];
      j=SNGL_SCAN[coeff_ctr][1];
    }

    //run++;
    // ilev=0;

    if(lossless_qpprime)
      level = absm (img->m7[j][i]);
    else
      level = (absm (m4[j][i])*levelscale[i][j] + leveloffset[i][j]);

    if (img->AdaptiveRounding)
    {
      if (lossless_qpprime || level == 0 )
      {
        img->fadjust4x4[intra][block_y+j][block_x+i] = 0;
      }
      else
      {
        img->fadjust4x4[intra][block_y+j][block_x+i] =
          (AdaptRndWeight * (absm(m4[j][i]) * levelscale[i][j] -
(level << q_bits)) + (1<< (q_bits))) >> (q_bits + 1);
      }
    }

      // we store the coefficients.
      c16[coeff_ctr]=level;
   }
```

At the code exposed before, "*Level*" is the absolute value of the quantized coefficients.

It is also interesting how the absolute values (levels) of the coefficients are stored in the *c16*. Note that the coefficients stored are those extracted in "DCT_LUMA", so only the luma coefficients are stored (see Figure 2.3).
These coefficients are stored because they have to be later modified by watermarking.

6.2.2. Generate the random errors

In order to obtain realistic results in the simulations, some Bit Error Rate (BER) is introduced inside the bitstream. The BER applied is a parameter introduced

at the initial configuration, so in encoder.cfg. Several simulations were performed using different values of BER (BER = {10^{-4}, 10^{-5}, 10^{-6}}).
A random bit error is generated for every recently coded slice (packet). Therefore, just after the slice has been encoded, the number of bits used by this slice can be extracted by looking at the following variable: "`img->currentSlice->partArr->bitstream->byte_pos`". So a random bit error rate is generated for every bit depending on the introduced parameter BER.

In this process of error generating, the number of possible errors inside this slice are counted. Then, other random processes take place in order to force the error inside a MB and the number of affected coefficients inside the block.

This way, it is possible to know if the error will be detected or not, depending on the watermarking applied at this MB.

Just in this part of the cycle, a file is generated where the following information is written:

1) Number of MB's affected, in order to be modified.
2) Number of the frame where the affected MB is located.
3) Number of the slice where the affected MB is.
4) And finally, it also says if the error is detected, in which case concealment method will be applied, or the opposite, the error is not detected and as a consequence this MB will be printed green, as it can bee seen in Figure 5.14.

With this information, the decoder can fulfil its functions in order to decode every MB for every simulation.

6.2.3. Inserting watermarking

Watermarking technique is applied in next step, only to the stored coefficients (c16 variable).

6.2.3.1. Inserting Relation Based Watermarking scheme

In the process of applying RBW for every block of the bitstream, the first function is adding all the *n* first coefficients of the block (see the equation 5.5).

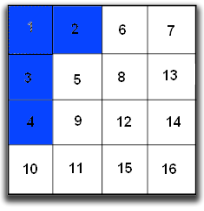


Figure 6.2: *Example of n=4 coefficients selected.*

```
// Are added the first n coefficients of a block

for (coeff_ctr=0;coeff_ctr<num_watermark;coeff_ctr++)
      sum_coeffs += c16[coeff_ctr];
```

Note that variable “sum_coeffs” is the sum of the *n* coefficients, also note that num_watermark represents the variable *n* in terms of the proposed equation.

Now, the coefficients are ordered from the lowest to the highest value, thus distortion is minimized because only the lowest coefficients will be modified.

```
for(i1=num_watermark-2;i1>-1;i1--)
{
      for (coeff_ctr=0;coeff_ctr < num_watermark;coeff_ctr++)
      {
            for(j1=14;j1>0;j1--)
            {
                  if(c_selec[j1]==coeff_ctr)
                  {
                        mark=1;
                        break;
                  }
            }
            if((!mark)&&(coeff_ctr!=0))
            {
                  if(max_coef<c16[coeff_ctr])
                  {
                        max_coef=c16[coeff_ctr];
                        c_selec[i1]=coeff_ctr;
                  }
            }
            mark=0;
      }
      max_coef=-1;
}
```

Here, variable c_selec stores the coefficients to be modified (*n*) from the lowest to the highest.

The result of applying modulo mod is obtained by using the following equation. Note that *mod_x*, is this variable mod regarding the proposed equation.

```
result_modulo = sum_coeffs%mod_x;
```

And then, the result of this equation must be forced to one *k* value (in terms of the equation), so value *k* is forced to “0” in order to make it easier to understand; the election of the value of *k* is independent, but it must be the same value during all the simulation.

Next, the sum of the coefficients is forced to be *k*, as the proposed equation shows. This will be performed as it is explained in Section 5.1.

```
if(result_modulo!=0)
{
      if(result_modulo>=(mod_x/2))
      {
            dif_modulo=mod_x-result_modulo;
            for (i1=0;i1<dif_modulo;i1++)
            {
              c16[c_selec[i1]]=c16[c_selec[i1]]++;
            }

      }
      else
      {
            for (i1=0;i1<result_modulo;i1++)
            {
                  j1=i1;
                  while(mark)
                  {
                        if(j1==num_watermark-1)
                        j1=0;
                        if (c16[c_selec[j1]]!=0)
                        {
                              mark=0;
                              break;
                        }

                        mark++;

                        if((mark-1)==num_watermark-1)
                        {
                             dif_modulo=mod_x-res_modulo;
                             for (i1=0;i1<dif_modulo;i1++)
                              {
                                 if(num_watermark-1<=i1)
                                   c16[c_selec[0]]=c16[c_selec[0]]++;
                                 else
                                   c16[c_selec[i1]]=c16[c_selec[i1]]++;
                              }
                              break;
                        }
                        j1++;
                  }
                  if(mark-1==num_watermark-1)
                        break;
                  mark=0;
                  i1=j1;
                  c16[c_selec[i1]]--;
            }
      }
}
```

Finally, the coefficients of a block are forced to fulfil the predefined equation, so after here, the sum of the coefficients makes 0 modulo *mod.*

Once the coefficients are watermarked (their values modified), the encoder continues operating in order to finish the cycle. After that, the encoder will begin the same cycle again with next macroblock. The encoder keeps doing this process (this cycle) until the last MB of the last frame, so until the entire sequence is encoded.

6.3. The decoder

On the other hand, the code was less modified at the decoder than at the encoder. Only one function is added, which decides what to do to the analyzed MB, if applying concealment, painting it green or not modifying it.

As it is explained in Section 6.1, Table 6.2 shows the basic information to initiate the decoding process.

Table 6.2: *Important values changed in the configuration file of the decoder.*

1	=	NAL mode (0=Annex B, 1: RTP packets)
3	=	Error Concealment method (0:Weight Av,1:Direct Interp,2:MV Interp,3:Copy
TXT_FILE	=	File writed in txt by the encoder, where is described the information about every macroblock.

In the simulations, the decoder obtains a .TXT file from the encoder. This file contains the necessary information to decode every MB.

The first step is recovering the information written at the encoder's side, as shown at the following code:

```
while (!feof(files->read_err))
{
   fscanf(files->read_err,"%d %d %d %d \n", &loc_err[i][0],
&loc_err[i][1],&loc_err[i][2],&loc_err[i][3]);
      i++;
}
```

Note that variable *files->read_err* represents the file number made at encoder. This information is stored in a new variable (a matrix): loc_err[ ][ ].
So Table 6.3 shows the information stored at variable loc_err[ ][ ].

Table 6.3: *Information stored at loc_err [ ] [ ] by the code added at the decoder.*

Loc_err [i] [x] :
[i] = number of the MB.

And then, inside file *image.c* there is function "*decode_one_slice"* which has the following code:

```
mark=0
for(scn_err=0;((loc_err[scn_err+1][1]!=0)&&(loc_err[scn_err][1]<=img-
>number));scn_err++)
{
      if((img->current_mb_nr==loc_err[scn_err][0])&&(img-
>number==loc_err[scn_err][1]))
      {
            if((loc_err[scn_err][3]==0))
                  mark=3;
            else
                  mark=8;
            break;
      }
}
```

Note that this code is inside the cycle that decodes the slice, so at this point it is analyzing MB by MB.

This code looks at every MB to check if it is in the list of MB's stored in the variable loc_err [ ] [ ]. So, as it can be seen in the code, it tries to find out if the number of MB [0] and the number of frame [1] are the same. If the information coincides, it looks at the fourth parameter [3]:

- If the MB affected, the fourth parameter of the txt file [3] is equal to 0; this means that this MB **was** detected, and as a consequence it is marked by 3 which means that concealment method (copy paste) will be applied to this MB.

- Another possibility is that the fourth parameter [3] is equal to 1, what means that this analyzed MB **was not** detected, and as a consequence it is marked with an 8, being therefore decoded as a green MB.

And finally, next code shows how every MB is decoded depending on its marks (No marks, mark = 3 or mark =8)

```
inp->conmeth=mark;

if (mark ==0)
      decode_one_macroblock(img,inp);
else
{
      switch(inp->conmeth)
      {
            case 0:
                  weightedav(img->current_mb_nr);
            break;
            case 1:
                  dirinter(img->current_mb_nr);
            break;
            case 2:
                  copyplusmv(img);
            break;
            case 3:
                  copy_paste(img);
            break;
            case 4:
                  copy_shift(img);
            break;
            case 5:
                  copy_shift_bc(img);
            break;
            case 6:
                  if (img->type == 2)  //if image is I type
                        copy_paste(img);
                  else
                        dec_wo_res(img); //deco
      de_one_macroblock(img,inp);
                                    //dec_wo_color_res(img,inp);
            break;
            case 7:
                  img->mb_data[img->current_mb_nr].ei_flag=1;
            break;
            case 8:
            break;
            default:
                  snprintf(errortext, ET_SIZE, "Error concealment
                           method %i is not supported", inp->conmeth);
                  error(errortext,400);
      }
}
```

As the previous code shows, if the mark is zero, the process continues working as usual, decoding the macroblock being analyzed at the moment. If the mark is different to 0, the code hops to function “decode_one_macroblock”, and this macroblock continues the process depending on the mark. However, note that for a mark = 8, it does not do any different process and this MB will become a green MB (which means that no errors were detected in it). The green colour was used to avoid the crash of the decoder in case of the VLC

desynchronisation. An alternative would be to apply a straight decoding as described in [16]. However, the distortion would be similar, since straight decoding of the desynchronised video stream causes annoying rectangular artefacts.

SECTION 7. CONCLUSIONS.

The aim of this thesis was to investigate a fragile forced even watermarking scheme for detection of the errors in the VLC stream of an H.264/AVC encoded video stream with CIF resolution. An alternative approach to FEW was proposed which turned out to provide considerably better results in terms of quality at the decoder, for the same rate at the encoder, after the watermarking was applied. The proposed approach is based on the forced relation between the chosen transformation domain coefficients. A combined method using RBW and VLC resynchronization information is further proposed, which even more improves the PSNR at the decoder. The modification of the transform coefficients can be chosen in order to minimize distortion. The choice of the extracting function and the number of coefficients to be watermarked determines the trade-off between the detection probability and distortion at the encoder. They should be chosen carefully according to the bit error probability of the transmission environment.

In order to evaluate the robustness of the two different schemes: force even watermarking and relation based watermarking, PSNR at the decoder and at the encoder was evaluated as well as the error detection probability.The new proposed method, RBW, notably improves robustness of the decoder in an error prone environment compared to the already existing method (FEW), both using the technique of fragile watermarking. The RBW shows a higher detection probability than FEW. Using RBW provides more than 90% of detection probability depending on two factors: the number of selected coefficients *n*, and the modulo *M* chosen. FEW can obtain a high range of probabilities, with an average between 50% and 60%. So, the detection probability can be as low as 5% for a high value of QP (around 28 or 30) and also greater watermarking starting position (*pos*). For low *pos* values (2 to 4), FEW achieves its maximum detection probability, which is not higher than 90% and results in considerable quality distortion.

The video stream at the encoder suffers lower quality degradation when using the new method proposed – RBW. The quality degradation measured by Y-PSNR is even 1 dB worse for FEW. The quality at the encoder is for the RBW again given by the two parameters – the modulo *M* and the number of coefficients affected *n*. Basically, a tradeoff between the robustness and the quality degradation has to be found. The appropriate choice of these parameters has to be based on the desired application, especially the average packet error probability of the transmission channel.

First, distortion at the decoder must be analyzed and studied together with the distortion at the encoder and error detection probability (which have already been analyzed). Using the results of these analyses, the most suitable parameters can be chosen in order to provide both, best quality and robustness.

RBW performs considerably better than FEW. Using RBW as a detection mechanism encoded with a QP of 16, the gain obtained is from 5 to 10 dB

higher than if no detection mechanism is applied (black line), and using QP of 28 to encode the same sequence that gain goes from 14 dB to 18 dB, always depending on the coefficients selected and on the modulo chosen to fulfil the equation.

Therefore, applying RBW as a method for error detection, will definitely improve the quality at the decoder working in an error prone environment. This makes the proposed method especially suitable for the application in the wireless mobile networks like UMTS, not only for the streaming, but also for the conversational services.

REFERENCES

[1] B.Sklar, *"Raleigh fading channels in mobile digital communication systems, part I: Charateritzation,"*. IEEE Communications, Mag, vol. 35, no. 9., (1997).

[2] Chen M, He Y., "*A fragile watermark error detection scheme for wireless video communications*", IEEE Transactions on multimedia, vol. 7, no. 2 (2005).

[3] H.264/AVC Software Coordination, "*Joint Model Software*", ver. 7.3, (available in http://iphome.hhi.de/suehring/tml/).

[4] O. Nemethova, G.C.Forte, M.Rupp, Robust Error Detection for H.264/AVC Using Relation Based Fragile Watermarking, Int. IEEE/EURASIP Conf. on Systems, Signals and Image Processing (IWSSIP), Budapest, Hungary, Sep. 2006.

[4] ITU-T and ISO/IEC JTC 1, "*Advanced video coding for generic audiovisual services*" - ITU-T Recommendation H.264 – ISO/IEC 28017 (2005)

[5] ITU-T and ISO/IEC JTC 1, "*Advanced video coding for generic audiovisual services*" - ITU-T Recommendation H.264 – ISO/IEC 28017 (2005)

[6] Information Technology, "*Coding of Moving Pictures and Associated Audio for Storage Media Upto 1.5 Mb/s"*. Part 2. Video, ISO/IEC DIS 11172-2, (1991).

[7] Information Technology — *"Generic Coding of Moving Pictures and Associated Audio".* Part 2. Video, ISO/IEC 13818-2, (1994).

[8] ISO/IEC JTC1/SC29/WG11 14496-2, "*Amd X, Coding of Moving Pictures and Audio*," International Standard, Maui, HI, (1999).

[9] ITU-T Rec. H.264 / ISO/IEC 11496-10, "*Advanced Video Coding*", Final Committee Draft, Document JVTE022, (2002).

[10] Joint Video Team of ITU-T and ISO/IEC JTC 1, "*Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification (ITU-T Rec. H.264 | ISO/IEC 14496-10 AVC),*" document JVT-G050r1, May 2003; technical corrigendum 1 documents JVT-K050r1 (non-integrated form) and JVT-K051r1 (integrated form), March 2004; and Fidelity Range Extensions documents JVT-L047 (non-integrated form) and JVT-L050 (integrated form), (July 2004).

[11] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "*Overview of the H.264/AVC Video Coding Standard*", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 13, no. 7, (2003).

[12] Richardson I., "*Prediction of Intra Macroblocks*", www.vcodex.com/h264.html (2002).

[13] Richardson I.,"*Prediction of Inter Macroblocks in P-slices.*", www.vcodex.com/h264.html (2002).

[14] Richardson I.,"*Transform and quantization*.", www.vcodex.com/h264.html (2002).

[15] Richardson I.,"*H.264 Variable Length Coding*", www.codex.com/h264.html (2002).

[16] L. Superiori, O. Nemethova, M. Rupp, "*Performance of a H.264/AVC Error Detection Algorithm Based on Syntax Analysis*" Int. Conf. on Advances in Mobile Computing and Multimedia (MoMM), Yogyakarta, Indonesia, (Dec. 2006).

[17] RFC 3828: "The lightweight user datagram protocol (udp-lite)", (July 2004).

[18] M.Barni, F.Bartolini, and P. Bianco, "*On the performance of syntax-based error detection in H.263 video coding: a quantitative analysis,"* in Electronic Imaging 2000-SPIE Conf. Image and Video Communications, San Jose, CA, (Jan. 2000).

[19] O. Nemethova, W. Karner, A. Al Moghrabi, M. Rupp
"Cross-Layer Error Detection for H.264 Video over UMTS", in Proc. Proceedings of the International Wireless Summit (WPMC 2005), Aalborg, Denmark, (September, 2005).

[20] O. Nemethova, J.C. Rodriguez, M. Rupp
"Improved Detection for H.264 Encoded Video Sequences over Mobile Networks", in Proc. Proceedings of the 8th International Symposium on Communication Theory and Applications, pages 343 - 348, Ambleside, Lake District, UK, (July, 2005).

[21] F. Hartung and B.Girod, "*Digital watermarking of MPEG 2 coded video in bitstream domain.*" IEEE ICASSP, (1997).

[22] Chen M, He Y., "*A fragile watermark error detection scheme for wireless video communications*", IEEE Transactions on multimedia, vol. 7, no. 2 (2005).

[23] JVT, "*Draft ITU-T recommendation and final draft international standard of joint video specification (ITU-T rec. H.254 – ISO/IEC 14496-10 AVC),*" (2003), JVT-G050 available on http://bs.hhi.de/˜wiegand/JVT.html.

[24] F. Bellifemine et all., "*Statistics analysis of the 2D-DCT coefficients of the differential signal of images*," Signal Processing: Image Communic., vol. 4, (1992).

[25] H.264/AVC Software Coordination, "Joint Model (JM) Reference Software," ver.10.1 (FREext), available in http://iphome.hhi.de/suehring/tml/.

ANNEXOS

TÍTOL DEL TFC/PFC

TITULACIÓ:

AUTOR: Gonzalo Calvar Forte

DIRECTOR: Olivia Nemethova

DATA: 25 de febrer de 2003

A Annex

A.1 List of abrevations

AVC	Advanced Video Coding
B frame	Bi-predictive frame
BEP	Bit Error Probability
BER	Bit Error Rate
BSC	Binary Symmetric Channel
CABAC	Context-Adaptative Binary Coding
CAVLC	Context Adaptive VLC
CIF	Common Intermediate Format
CRC	Cyclic Redundancy Check
DCT	Discrete Cosine Transform
FEC	Forward Error Control
FEW	Force Even Watermarking
I frame	Intra-predicted frame
IDCT	Inverse DCT
IDR	Instantaneous Decoder Refresh
IEC	International Electrotechnical Commission
IP	Internet Protocol
ISO	International Standards Organization
ITU	International Telecommunication Union
ITU-T	ITU Telecommunication Standardization Sector
JVT	Joint Video Team
MB	Macro Block
MMS	Multimedia Messaging System
MP4	MPEG-4 Part 14
MPEG	Moving Picture Experts Group
MSE	mean square error
MTU	Maximum Transport Unit
NAL	Network Adaptation Layer
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
P frame	Predictive frame
PDU	Protocol Data Unit
PSNR	Peak (Pixel) Signal to Noise Ratio
RBW	Relation Based Watermarking
RLC	Radio Link Control
RTP	Real-time Transport Protocol
QP	Quantized Parameter
QVGA	Quarter Video Graphics Array
VCEG	Video Coding Experts Group
VCL	Video Coding Layer
VLC	Variable Length Code

A.2 Encoder configuration file : encoder.cfg

```
# New Input File Format is as follows
# <ParameterName> = <ParameterValue> # Comment
#
# See configfile.h for a list of supported ParameterNames


###########################################################################
# Files
###########################################################################I
InputFile                   =     "carphone_cif.yuv"      # Input sequence
InputHeaderLength           = 0       # If the inputfile has a header, state it's
                                      #length in byte here
StartFrame                  =  0      # Start frame for encoding. (0-N)
FramesToBeEncoded           =  250    # Number of frames to be coded
FrameRate                   = 30.0    # Frame Rate per second (0.1-100.0)
SourceWidth                 = 352     # Frame width
SourceHeight                = 288     # Frame height
TraceFile                   = "trace_enc.txt"
ReconFile                   = "test_rec.yuv"
OutputFile                  = "stream_output.264"

###########################################################################
# Encoder Control
###########################################################################
ProfileIDC                  =  100    # Profile IDC (66=baseline, 77=main,
                                      # 88=extended; FREXT Profiles: 100=High,
                                      # 110=High 10, 122=High 4:2:2,144=High
                                      # 4:4:4, for params see below)
LevelIDC                    = 40      # Level IDC   (e.g. 20 = level 2.0)
IntraPeriod                 =  0      # Period of I-Frames (0=only first)
EnableOpenGOP               = 0       # Support for open GOPs (0: disabled, 1:
                                      #enabled)
IDRIntraEnable              =  0      # Force IDR Intra  (0=disable 1=enable)
QPISlice                    = 22      # Quant. param for I Slices (0-51)
QPPSlice                    = 22      # Quant. param for P Slices (0-51)
FrameSkip                   =  0      # Number of frames to be skipped in input
                                      # (e.g 2 will code every third frame)
ChromaQPOffset              =  0      # Chroma QP offset (-51..51)
UseHadamard                 =  1      # Hadamard transform (0=not used, 1=used
                                      # for all subpel positions, 2=use only for
                                      # qpel)
DisableSubpelME             = 0       # Disable Subpixel Motion Estimation
                                      # (0=off/default, 1=on)
SearchRange                 = 16      # Max search range
NumberReferenceFrames       =  1      # Number of previous frames used for inter
                                      #motion search (1-16)
PList0References            =  0      # P slice List 0 reference override (0
                                      #disable, N <= NumberReferenceFrames)
```

```
Log2MaxFNumMinus4       = 0     # Sets log2_max_frame_num_minus4
                                #(-1 :based on FramesToBeEncoded/Auto,
                                #>=0 : Log2MaxFNumMinus4)
                                #Log2MaxPOCLsbMinus4       = -1
                                #Sets log2_max_pic_order_cnt_lsb_minus4
                                #(-1 : Auto, >=0 : Log2MaxPOCLsbMinus4)

GenerateMultiplePPS     =  0    # Transmit multiple parameter sets.
                                #Currently parameters basically enable all
                                #WP modes (0: diabled, 1: enabled)
ResendPPS               = 0     # Resend PPS (with pic_parameter_set_id
#0) for every coded Frame/Field pair (0: disabled, 1: enabled)

MbLineIntraUpdate       =  0    # Error robustness(extra intra macro block
                                # updates)(0=off, N: One GOB every N
                                #frames are intra coded)
RandomIntraMBRefresh    = 0     # Forced intra MBs per picture
InterSearch16x16        = 1     # Inter block search 16x16 (0=disable,
                                #1=enable)
InterSearch16x8         = 0     # Inter block search 16x8  (0=disable,
                                #1=enable)
InterSearch8x16         = 0     # Inter block search  8x16 (0=disable,
                                #1=enable)
InterSearch8x8          = 0     # Inter block search  8x8  (0=disable,
                                #1=enable)
InterSearch8x4          =  0    # Inter block search  8x4  (0=disable,
                                #1=enable)
InterSearch4x8          =  0    # Inter block search  4x8  (0=disable,
                                #1=enable)
InterSearch4x4          =  0    # Inter block search  4x4  (0=disable,
                                #1=enable)

IntraDisableInterOnly   = 0     # Apply Disabling Intra conditions only to
                                #Inter Slices (0:disable/default,1: enable)
Intra4x4ParDisable      = 0     # Disable Vertical & Horizontal 4x4
Intra4x4DiagDisable     = 0     # Disable Diagonal 45degree 4x4
Intra4x4DirDisable      = 0     # Disable Other Diagonal 4x4
Intra16x16ParDisable    = 0     # Disable Vertical & Horizontal 16x16
Intra16x16PlaneDisable  = 0     # Disable Planar 16x16
ChromaIntraDisable      = 0     # Disable Intra Chroma modes other than
                                #DC
DisposableP             = 0     # Enable Disposable P slices in the primary
                                #layer (0: disable/default, 1: enable)
DispPQPOffset           = 0     # Quantizer offset for disposable P slices (0:
                                #default)

###################################################################
# B Slices
###################################################################
```

```
NumberBFrames                = 0    # Number of B coded frames inserted
                                    #(0=not used)
QPBSlice                     = 30   # Quant. param for B slices (0-51)
BRefPicQPOffset              =  0   # Quantization offset for reference B coded
                                    #pictures (-51..51)
DirectModeType               =  1   # Direct Mode Type (0:Temporal 1:Spatial)
DirectInferenceFlag          =  1   # Direct Inference Flag (0: Disable 1:
                                    #Enable)
BList0References             =  0   # B slice List 0 reference override (0
                                    #disable, N <= NumberReferenceFrames)
BList1References             =  1   # B slice List 1 reference override (0
                                    #disable, N <= NumberReferenceFrames)
                                     # 1 List1 reference is usually recommended
                                    # for normal GOP Structures.
                                    # A larger value is usually more appropriate
                                    # if a more flexible structure is used (i.e.
                                    # using PyramidCoding)

BReferencePictures           =  0   # Referenced B coded pictures (0=off,
                                    #1=on)

PyramidCoding                = 0    # B pyramid (0= off, 1= 2 layers, 2= 2 full
                                    #pyramid, 3 = explicit)
PyramidLevelQPEnable         = 1    # Adjust QP based on Pyramid Level (in
                                    #increments of 1). Overrides
                                    # BRefPicQPOffset behavior.(0=off, 1=on)
ExplicitPyramidFormat        = "b2r28b0e30b1e30b3e30b4e30"
                                    # Explicit Enhancement GOP. Format is
                                    #{FrameDisplay_orderReferenceQP}.
                                     #Valid values for reference type is
                                    #r:reference, e:non reference.
PyramidRefReorder            =  1    # Reorder References according to Poc
                                    #distance for PyramidCoding (0=off,
                                    #1=enable)
PocMemoryManagement          = 1    # Memory management based on Poc
                                    #Distances for PyramidCoding (0=off, 1=on)
BiPredMotionEstimation       = 0    # Enable Bipredictive based Motion
                                    #Estimation (0:disabled, 1:enabled)
BiPredMERefinements          = 3    #Bipredictive ME extra refinements (0:
                                    #single, N: N extra refinements (1 default)
BiPredMESearchRange          = 16   # Bipredictive ME Search range (8 default).
                                    #Note that range is halved for every extra
                                    #refinement.
BiPredMESubPel               = 1    # Bipredictive ME Subpixel Consideration
                                    #(0: disabled, 1: single level, 2: dual level)
```

```
###################################################################
# SP Frames
###################################################################
SPPicturePeriodicity      = 0     # SP-Picture Periodicity (0=not used)
QPSPSlice                 = 28    # Quant. param of SP-Slices for Prediction
                                  #Error (0-51)
QPSP2Slice                = 27    # Quant. param of SP-Slices for Predicted
                                  #Blocks (0-51)

###################################################################
# Output Control, NALs
###################################################################

SymbolMode                = 0     # Symbol mode (Entropy coding method:
                                  # 0=UVLC, 1=CABAC)
OutFileMode               =  1    # Output file mode, 0:Annex B, 1:RTP
PartitionMode             =  0    # Partition Mode, 0: no DP, 1: 3 Partitions
                                  #per Slice

###################################################################
# CABAC context initialization
###################################################################

ContextInitMethod         =  1    # Context init (0: fixed, 1: adaptive)
FixedModelNumber          =  0    # model number for fixed decision for inter
                                  #slices ( 0, 1, or 2 )

###################################################################
# Interlace Handling
###################################################################

PicInterlace              = 0     # Picture AFF    (0: frame coding, 1: field
                                  #coding, 2:adaptive frame/field coding)
MbInterlace               = 0     # Macroblock AFF (0: frame coding, 1: field
                                  #coding, 2:adaptive frame/field coding)
IntraBottom               =  0    # Force Intra Bottom at GOP Period

###################################################################
# Weighted Prediction
###################################################################

WeightedPrediction        = 0     # P picture Weighted Prediction (0=off,
                                  #1=explicit mode)
WeightedBiprediction      = 0     # B picture Weighted Prediciton (0=off,
                                  #1=explicit mode,  2=implicit mode)
UseWeightedReferenceME= 0         # Use weighted reference for ME (0=off,
                                  #1=on)
```

```
##################################################################
# Picture based Multi-pass encoding
##################################################################

RDPictureDecision          =  0     # Perform RD optimal decision between
                                    #different coded picture versions.
                                    # If GenerateMultiplePPS is enabled then
                                    #this will test different WP methods.
                                    # Otherwise it will test QP +-1 (0: disabled,
                                    #1: enabled)
RDPictureIntra             =  0     # Perform RD optimal decision also for intra
                                    #coded pictures (0: disabled (default), 1:
                                    #enabled).
RDPSliceWeightOnly         =  1     # Only consider Weighted Prediction for P
                                    #slices in Picture RD decision. (0: disabled,
                                    #1: enabled (default))
RDBSliceWeightOnly         = 0      # Only consider Weighted Prediction for B
                                    #slices in Picture RD decision. (0: disabled
                                    #(default), 1: enabled )

##################################################################
# Loop filter parameters
##################################################################

LoopFilterParametersFlag   = 0      # Configure loop filter (0=parameter below
                                    #ingored, 1=parameters sent)
LoopFilterDisable          = 0      # Disable loop filter in slice header (0=Filter,
                                    #1=No Filter)
LoopFilterAlphaC0Offset    = 0      # Alpha & C0 offset div. 2, {-6, -5, ... 0, +1, ..
                                    #+6}
LoopFilterBetaOffset       = 0      # Beta offset div. 2, {-6, -5, ... 0, +1, .. +6}

##################################################################
# Error Resilience / Slices
##################################################################

SliceMode                  = 2      # Slice mode (0=off 1=fixed #mb in slice
                                    #2=fixed #bytes in slice 3=use callback)
SliceArgument              = 1200   # Slice argument (Arguments to modes 1
                                    #and 2 above)

num_slice_groups_minus1    = 0      # Number of Slice Groups Minus 1, 0 == no
                                    #FMO, 1 == two slice groups, etc.
slice_group_map_type       = 0      # 0:  Interleave, 1: Dispersed,
                                    #2: Foreground with left-over,
                                    # 3:  Box-out,    4: Raster Scan   5: Wipe
                                    # 6:Explicit, slice_group_id read from
                                    #                 SliceGroupConfigFileName
slice_group_change_direction_flag = 0  # 0: box-out clockwise, raster scan or
                                       #wipe right,
```

```
slice_group_change_rate_minus1  = 85
SliceGroupConfigFileName        = "sg0conf.cfg" #Used for
                                  # slice_group_map_type 0, 2, 6
UseRedundantSlice       = 0     # 0: not used, 1: one redundant slice used
                                  # for each slice (other modes not supported
                                  #yet)

##########################################################################
# Search Range Restriction / RD Optimization
##########################################################################

RestrictSearchRange     =  2    # restriction for (0: blocks and ref, 1: ref, 2:
                                  #no restrictions)
RDOptimization          =  0    # rd-optimized mode decision
                                  # 0: RD-off (Low complexity mode)
                                  # 1: RD-on (High complexity mode)
                                  # 2: RD-on (Fast high complexity mode - not
                                  #work in FREX Profiles)
                                  # 3: with losses
DisableThresholding     = 0     # Disable Thresholding of Transform
                                  #Coefficients (0:off, 1:on)
DisableBSkipRDO         = 0     # Disable B Skip Mode consideration from
                                  #RDO Mode decision (0:off, 1:on)
SkipIntraInInterSlices  = 0     # Skips Intra mode checking in inter slices if
                                  #certain mode decisions are satisfied (0: off,
                                  #1: on)
                                  # Explicit Lambda Usage
UseExplicitLambdaParams=0       # Use explicit lambda scaling parameters
                                  #(0:disabled, 1:enabled)
LambdaWeightIslice      =  0.65 # scaling param for I slices.
LambdaWeightPslice      = 0.68  # scaling param for P slices.
LambdaWeightBslice      =  2.00 # scaling param for B slices.
LambdaWeightRefBslice   =  1.50 # scaling param for Referenced B slices.
LambdaWeightSPslice     =  1.50 # scaling param for SP slices.
LambdaWeightSIslice     = 0.65  # scaling param for SI slices.
LossRateA               = 0     # expected packet loss rate of the channel
                                  # for the first partition, only valid if
                                  # RDOptimization = 2
LossRateB               = 0     # expected packet loss rate of the channel
                                  #for the second partition, only valid if
                                  #RDOptimization = 2
LossRateC               = 0     # expected packet loss rate of the channel
                                  #for the third partition, only valid if
                                  #RDOptimization = 2
NumberOfDecoders        = 30    # Numbers of decoders used to simulate the
                                  #channel, only valid if RDOptimization = 2
RestrictRefFrames       = 0     # Doesnt allow reference to areas that have
                                  #been intra updated in a later frame.
```

```
##################################################################
# Additional Stuff
##################################################################

UseConstrainedIntraPred  = 0      # If 1, Inter pixels are not used for Intra
                                  #macroblock prediction.
LastFrameNumber          =  0     # Last frame number that have to be coded
                                  #(0: no effect)
ChangeQPI                = 16     # QP (I-slices)  for second part of sequence
                                  #(0-51)
ChangeQPP                = 16     # QP (P-slices)  for second part of sequence
                                  #(0-51)
ChangeQPB                = 18     # QP (B-slices)  for second part of sequence
                                  #(0-51)
ChangeQPBSRefOffset      =  2     # QP offset (stored B-slices)  for second
                                  #part of sequence (-51..51)
ChangeQPStart            =  0     # Frame no. for second part of sequence (0:
                                  #no second part)

NumberofLeakyBuckets     =  8     # Number of Leaky Bucket values
LeakyBucketRateFile      =  "leakybucketrate.cfg"
                                  # File from which encoder derives rate
                                  #values
LeakyBucketParamFile     =  "leakybucketparam.cfg"
                                  # File where encoder stores
                                  # leakybucketparams

NumberFramesInEnhancementLayerSubSequence  = 0  # number of frames in
                                                #the Enhanced Scalability
                                                # Layer
NumberOfFrameInSecondIGOP = 0     # Number of frames to be coded in the
                                  #second IGOP

SparePictureOption       =  0     # (0: no spare picture info, 1: spare picture
                                  #available)
SparePictureDetectionThr  = 6     # Threshold for spare reference pictures
                                  # detection
SparePicturePercentageThr =92     #Threshold for the spare macroblock
                                  # percentage.

PicOrderCntType           = 2     # (0: POC mode 0, 1: POC mode 1, 2: POC
                                  #mode 2)

##################################################################
#Rate control
##################################################################

RateControlEnable        =  0      # 0 Disable, 1 Enable
Bitrate                  =  45020 # Bitrate(bps)
InitialQP                = 24      # Initial Quantization Parameter for the first
```

```
                                    #I frame
BasicUnit                 = 11      # Number of MBs in the basic unit
                                    # should be a fractor of the total number
                                    # of MBs in a frame
ChannelType               = 0       # type of channel( 1=time varying channel;
                                    #0=Constant channel)

###################################################################
#Fast Mode Decision
###################################################################

EarlySkipEnable           = 0       # Early skip detection (0: Disable 1: Enable)
SelectiveIntraEnable      = 0       # Selective Intra mode decision (0: Disable
                                    #1: Enable)

###################################################################
#FREXT stuff
###################################################################

YUVFormat                 = 1       # YUV format (0=4:0:0, 1=4:2:0, 2=4:2:2,
                                    #3=4:4:4)
RGBInput                  = 0       # 1=RGB input, 0=GBR or YUV input
BitDepthLuma              = 8       # Bit Depth for Luminance (8...12 bits)
BitDepthChroma            = 8       # Bit Depth for Chrominance (8...12 bits)
CbQPOffset                = 0       # Chroma QP offset for Cb-part (-51..51)
CrQPOffset                = 0       # Chroma QP offset for Cr-part (-51..51)
Transform8x8Mode          = 0       # (0: only 4x4 transform, 1: allow using 8x8
                                    #transform additionally, 2: only 8x8
                                    #transform)
ResidueTransformFlag      = 0       # (0: no residue color transform 1: apply
                                    #residue color transform)
ReportFrameStats          = 0       # (0:Disable Frame Statistics 1: Enable)
DisplayEncParams          = 0       # (0:Disable Display of Encoder Params 1:
                                    #Enable)
Verbose                   = 1       #level of display verboseness (0:short,
                                    #1:normal, 2:detailed)

###################################################################
#Q-Matrix (FREXT)
###################################################################
QmatrixFile               = "q_matrix.cfg"
ScalingMatrixPresentFlag  = 0       # Enable Q_Matrix  (0 Not present, 1
                                    #Present in SPS, 2 Present in PPS, 3
                                    # Present in both SPS & PPS)
                                    #ScalingListPresentFlag0  = 3
                                    # Intra4x4_Luma    (0 Not present,
                                    #1 Present in SPS, 2 Present in PPS, 3
                                    #Present in both SPS & PPS)
ScalingListPresentFlag1   = 3       # Intra4x4_ChromaU (0 Not present, 1
                                    #Present in SPS, 2 Present in PPS,
```

```
                                #3 Present in both SPS & PPS)
ScalingListPresentFlag2    = 3  # Intra4x4_chromaV (0 Not present,
                                #1 Present in SPS, 2 Present in PPS,
                                #3 Present in both SPS & PPS)
ScalingListPresentFlag3    = 3  # Inter4x4_Luma    (0 Not present,
                                #1 Present in SPS, 2 Present in PPS,
                                #3 Present in both SPS & PPS)
ScalingListPresentFlag4    = 3  # Inter4x4_ChromaU (0 Not present,
                                #1 Present in SPS, 2 Present in PPS, 3
                                #Present in both SPS & PPS)
ScalingListPresentFlag5    = 3  # Inter4x4_ChromaV (0 Not present,
                                1 Present in SPS, 2 Present in PPS, 3
                                Present in both SPS & PPS)
ScalingListPresentFlag6  = 3    # Intra8x8_Luma    (0 Not present,
                                1 Present in SPS, 2 Present in PPS,
                                3 Present in both SPS & PPS)
ScalingListPresentFlag7  = 3    # Inter8x8_Luma    (0 Not present,
                                1 Present in SPS, 2 Present in PPS,
                                3 Present in both SPS & PPS)

###################################################################
#Rounding Offset control
###################################################################

OffsetMatrixPresentFlag    = 0  # Enable Explicit Offset Quantization
                                #Matrices  (0: disable 1: enable)
QOffsetMatrixFile       = "q_offset.cfg"  # Explicit Quantization Matrices file

AdaptiveRounding           = 0  # Enable Adaptive Rounding based on JVT-
                                #N011 (0: disable, 1: enable)
AdaptRndPeriod             = 1  # Period in terms of MBs for updating
                                #rounding offsets.
                                # 0 performs update at the picture level.
                                #Default is 16. 1 is as in JVT-N011.
AdaptRndChroma             = 0  # Enables coefficient rounding adaptation
                                #for chroma

AdaptRndWFactorIRef        = 4  # Adaptive Rounding Weight for I/SI slices
                                #in reference pictures /4096
AdaptRndWFactorPRef      = 4    # Adaptive Rounding Weight for P/SP slices
                                #in reference pictures /4096
AdaptRndWFactorBRef      = 4    # Adaptive Rounding Weight for B slices in
                                #reference pictures /4096
AdaptRndWFactorINRef     = 4    # Adaptive Rounding Weight for
                                #I/SI slices in non reference pictures /4096
AdaptRndWFactorPNRef     = 4    # Adaptive Rounding Weight for P/SP slices
                                #in non reference pictures /4096
AdaptRndWFactorBNRef     = 4    # Adaptive Rounding Weight for
                                #B slices in non reference pictures /4096
```

```
###################################################################
#Lossless Coding (FREXT)
###################################################################

QPPrimeYZeroTransformBypassFlag = 0    # Enable lossless coding when
                                       #qpprime_y is zero (0 Disabled, 1 Enabled)

###################################################################
#Fast Motion Estimation Control Parameters
###################################################################

UseFME                  = 0     # Use fast motion estimation
                                #(0=disable/default, 1=UMHexagonS,
                                # 2=Simplified UMHexagonS, 3=EPZS
                                # patterns)

EPZSPattern             = 2     # Select EPZS primary refinement pattern.
                                # (0: small diamond, 1: square, 2: extended
                                #diamond/default, 3: large diamond)

EPZSDualRefinement      = 3     # Enables secondary refinement pattern.
                                # (0:disabled, 1: small diamond, 2: square,
                                # 3: extended diamond/default, 4: large
                                #diamond)
EPZSFixedPredictors     = 2     # Enables Window based predictors
                                # (0:disabled, 1: P only, 2: P and B/default)
EPZSTemporal            = 1     # Enables temporal predictors
                                # (0: disabled, 1: enabled/default)
EPZSSpatialMem          = 1     # Enables spatial memory predictors
                                # (0: disabled, 1: enabled/default)
EPZSMinThresScale       = 0     # Scaler for EPZS minimum threshold
                                # Increasing value can speed up encoding.
EPZSMedThresScale       = 1     # Scaler for EPZS median threshold
                                # Increasing value can speed up encoding.
EPZSMaxThresScale       = 1     # Scaler for EPZS maximum threshold
                                # Increasing value can speed up encoding.
```

A.3 Decoder configuration file : decoder.cfg

```
stream_output.264       .....H.26L coded bitstream
output.yuv              .....Output file, YUV/RGB
carphone_cif.yuv        .....Ref sequence (for SNR)
1                       .....Write 4:2:0 chroma components for monochrome
                             streams
1                       .....NAL mode (0=Annex B, 1: RTP packets)
0                       .....SNR computation offset
2                       .....Poc Scale (1 or 2)
500000                  .....Rate_Decoder
104000                  .....B_decoder
73000                   .....F_decoder
leakybucketparam.cfg    .....LeakyBucket Params
0                       .....Err Concealment(0:Off,1:Frame Copy,2:Motion
                             Copy
2                       .....Reference POC gap (2: IPP (Default), 4: IbP / IpP)
2                       .....POC gap (2: IPP /IbP/IpP (Default), 4: IPP with
                             frame skip = 1 etc.)
3                       .....Error Concealment method (0:Weight Av,
                             1:Direct Interp,2:MV Interp,3:Copy Paste,
                             4:Boundary Match,5:Block Match,
                             6:Dec Without Res,7:Conc implem by H264,
                             8:No conc)
0                       .....Error rate per slice (%)
```

This is a file containing input parameters to the JVT H.264/AVC decoder.
The text line following each parameter is discarded by the decoder.