TECHNISCHE UNIVERSITÄT WIEN

CENTRO POLITÉCNICO SUPERIOR DE ZARAGOZA,

CPS

---

Institut für Nachrichtentechnik und Hochfrequenztechnik


Master Thesis

# ENCODING OPTIMIZATION OF H.264/AVC SOCCER VIDEO SEQUENCES

Professor: Markus RUPP

Supervisor: Luca SUPERIORI

Author: Alfredo FONT PEREZ

# Contents

# Abstract

Football video sequences are one of the most streamed contents over internet and in 3G networks, with special emphasis in mobile applications. Because of this, it is becoming more important the information compression and therefore the improvement of video standards. This work deals with the compression improvement of soccer sequences in the standard video H.264/AVC or MPEG-4 Part 10.

The main objective of this work was to optimize the codification efficiency in soccer video sequences with the H.264/AVC standard. The MBs corresponding to the audience are encoded with a significant amount of bits because they contain a very high frequency and they are difficult to predict. Because subjectively we think the audience is the least important part of a soccer video, we have thought about modifying its encoding. We have used the FMO slicing strategy in order to (after segmenting the frames into three different regions: field, players and audience) be able to encode each region independently and transmitted in different packets.

We have investigated the possibility of modifying its encoding (H.264/AVC encoder) in order to exploit some characteristics of the audience to be captured according to the camera. Exploiting the camera movement to predict a "Global motion vector", we can think about sending only one motion vector for the whole audience indicating the camera movement to copy MBs directly from the previous image, but shifted.

The first of all, we have studied the working of the encoding mainly at MB level. We have implemented a new encoding method (modifying functions and variables) able to encode all the macroblocks (MBs) as skip, but reconstructing them at the both sides (encoder and decoder) with one GMV (transmitted in the

first MB of the slice) used for the whole audience. In that way we will not send residuals.

We have worked in a simulated environment, videos have been preanalyzed previously in Matlab to obtain information to be input to the encoder. We have obtained the GMVs, border maps and zoom detection for all the videos. When zoom appears the method cannot be applied, so a zoom detector have been implemented, using the MVs distribution of the audience MBs in order to detect the zoom.

Some effects as the zoom detection reliability have to be taken into consideration. But, in general, after the application of the new encoding method, some significant bit rate results have been obtained. The PSNR as a static parameter does not reflect the perceived quality. Because of that we have prepared a subjective quality test (MOS) in a typical UMTS device in order to evaluate it. The results have denoted the viewers do not pay much attention to the audience, even the quality used decreases.

This method can be used in different applications in the future. In this work we have used the Visual C++ program to work with the encoder and the mathematical program Matlab to preanalyze the videos, to obtain parameters and to analyze the final results.

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivation

In the last years, multimedia transmissions have become more and more popular, until the point that nowadays we can watch a video live from our mobile phone. With the previous 2G systems, defined by the Global System for Mobile communications (GSM) [1], nobody could have imagined watching videos in the mobile phone was possible, but now it has become a reality. Since the appearance of the third mobile generation a few years ago, with the Universal Mobile Telecommunications System (UMTS) [2] as a successor of GSM in Europe, the mobile phone concept has changed radically. The three main characteristics of GSM were: multimedia capability, a good internet connection enabling high rates of audio and video transmission and a high quality voice transmission comparable with fixed telephony. Instead of being a simple communication device, the mobile phone has become a multimedia device with multiple capabilities for the leisure time due to the big amount of applications offered. Nowadays we have the possibility of a wide range of applications in our mobile phones, such video streaming, being a one-way quasi real time application, which requires a considerable amount of available bandwidth to be sent.

The amount of possible applications offered and the development of new services like multimedia transmissions, specially football video transmissions due to

its popularity, require a very high bandwidth needed to be sent due to the high volume of data to represent them. Because of that and because of the huge demand for these kinds of services sharing the limited radio bandwidth, the efficiency of the data transmission and the compression of the data sent is becoming very important. The development of new video standards continues being done constantly.

Due to the limited capacity, an essential thing to let multimedia transmissions work is to use efficient video standards to compress the video. A video codec is a device or software that enables video compression and/or decompression for digital video, an imperative necessity since digital communications have grown so fast in the last decade. Efficient but also lossy compression codecs are used, inducing quality degradation. One of the codecs used nowadays, which has a significant reduction in the bit rate, compared to the other standards for the similar degree of quality, is the H.264/AVC standard, that will be the standard used in this thesis.

This project addresses the possibility of exploiting some characteristics of the football video sequences to modify the encoding of H.264/AVC standard in order to decrease the amount of data sent. It presents the implementation of a new encoding method of one particular region of a football video sequence after using FMO (Flexible Macroblock Ordering), a slicing strategy from standard H.264/AVC, to segment the video in different zones. We will explain the objectives of the project in the section 1.4 after having explained the main working of the H.264 standard and the previous work made [14], in order to make easier the comprehension of the scopes.

This project requires modifications of the encoding of H.264/AVC codec making it understandable for the standard H.264/AVC decoder. It deals with the transmission of CIF (352x288 pixels resolution) football videos encoded with the H.264/AVC codec within a simulated environment. In the present project the videos will be preanalyzed with Matlab, a mathematical program, in order to obtain information to exploit the possibilities of the encoder.

## 1.2 H.264/AVC Overview

Before diving into the project, it is useful to know how video coding is performed. Readers that are already well versed in this subject may skip this particular section and go directly to section 1.3, in which the origin and previous work of this thesis is explained. Although this overview will be explained supposing a minimum video coding knowledge from the reader, it is recommended to read some previous concepts of prior H.264/AVC video standards, that will be commented right now.

Video standards have been developed with the objective of comprising a wide variety of applications, from digital storage to multimedia transmission. Video standards achieve a high compression using several methods that exploit the temporal and spatial redundancy. Moving Pictures Expert Group (MPEG) is the expert group of the International Organization for Standardization (ISO/IEC) charged of the development of video and audio codification. From the 90's, the video coding expert group (VCEG) of the international telecommunication union - telecommunication sector (ITU-T) and the MPEG of ISO/IEC, already mentioned before, focused their investigations in different video coding techniques for several applications. These two groups have started their deployments with different objectives, while VCEG developed the H.261 standard [3] for video-conferencing applications, MPEG developed the MPEG-1 standard [4] for video storage. Although MPEG is a lossy video standard usually MPEG videos have a higher quality than other formats because the lost suffered is usually imperceptive to the human eye, that recognizes much better the luminance (white and black levels) than the chrominance (colour level), so the perceived quality depends strongly on the first one, and that information is exploited by this standard.

After the creation of MPEG-2 standard [5] (main standard used for most of TV standards, including NTSC, satellite TV and cable TV) as an extension of MPEG-1, the ITU-T adopted it also as standard H.262 [6]. Due to the necessity of comprising a wider range of applications, the MPEG developed the standard MPEG-4 part 2 [7] including the possible segmentation of video objects. At the

same time the VCEG developed the standard H.263 [8] for video mobile applications, compatible with MPEG-4 part 2. At that point, the two groups decided to join their efforts to create an overall group together called joint video team (JVT) to work in a new and better video standard. As a result of the deployment emerged the standard H.264 or MPEG-4 part 10 [9], providing a better integration to protocols and architectures, which will be the standard used in the present project.

Each new improvement usually requires more computational capability to encode, but at the same time, offers more possibilities to encode videos in different ways and more compression capability and error concealment. For low-rate video streaming a very high compression rate has to be used. The standard H.264/MPEG-4 part 10, is not only efficient for video storage, but also provides a high performance in terms of compression and is more robust against errors than the previous MPEG standards, as mainly MPEG-2, H.263 and MPEG-4 Part 2, standards used in multimedia transmission. This standard improves the rate-distortion efficiency relative to prior standards and provides reduction in the bit rate up to 50% compared with other standards.

H.264 has the same main conceptual blocks that the prior MPEG standards. In this present section 1.2 we are going to see a general overview of the standard H.264 including some technical features and we will describe the coding algorithm focusing the more relevant parts related with this project, we are going to introduce the standard step by step.

### 1.2.1 Introducing H.264/AVC

The video is the result of the process of capturing, recording, storing and reconstructing a sequence of consecutive images that, properly visualized, create a motion perception. These images are called frames in technical video language. The video can be interlaced or progressive depending on the encoding of the frames, interlaced frames or progressive frames. Generally, a frame is partitioned into rows,

a video contains two interleaved fields, the top field containing the even rows and the bottom field containing the odd rows. If the fields of the frame have been captured at the same instant, the frame is called progressive (these frames have been captured in a frame period), otherwise the two fields have been captured separated in time by a field period (half of a frame period) and the frame is called interlaced. In the interlaced case the fields can be considered frames with half of the information of a normal frame and with a half capturing time also. We can see both cases in Figure 1.1. In this present thesis we will work with progressive video.

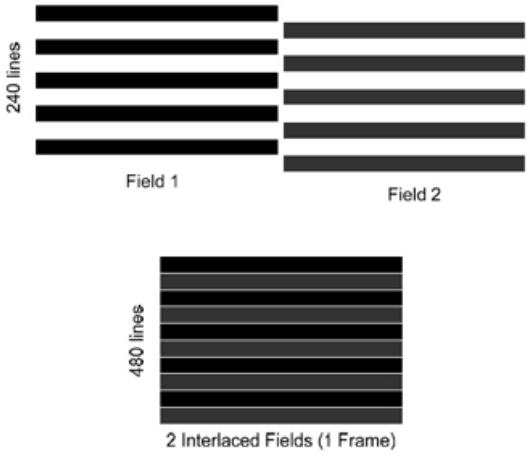


Figure 1.1: Interlaced and progressive frame.

Each frame is sampled with a fixed resolution, that determines the number of pixels in which is divided the capture. The more samples the video capture have, the more reliable is, but of course more bits have to be sent. In this project we will work with CIF resolution,in Table 1.1 the most common resolutions for mobile video are presented:

| Abbreviation | Size | Description |
|---|---|---|
| VGA | 640x480 | Video Graphics Array |
| QVGA | 320x240 | Quarter Video Graphics Array,<br>called also Standard Interchange Format (SIF) |
| Q2VGA | 160x120 | |
| CIF | 352x288 | Common Intermediate Format<br>(quarter of resolution 704x576 used in PAL) |
| QCIF | 176x144 | Quarter Common Intermediate Format |

Table 1.1: Most common resolutions for mobile video.

Since the human visual system is more sensitive to luminance than to color, a new system appeared to replace the RGB (Red, Green, Blue) color model to represent the pixels, the YUV color model. Instead of representing each pixel with three color components with RGB, the signal is divided into luminance (denoted as Y) and two color difference components (denoted as U and V). We are not going to show the conversion equations here, but this can be seen in: [10]. With YUV, we have also three components, but as we told before, the human eye is less sensitive to chroma than to luminance, so this can be exploited by the encoder. The accuracy of the luminance (brightness)has far more impact on the image discerned than that of the other two. Exploiting this human shortcoming, the codec can reduce the amount of data consumed by the chrominance considerably reducing the bandwidth. Therefore, the resulting U and V signals can be compressed. There are several modes to sample the video, originally each sample was encoded for each pixel, but nowadays the most common way of subsampling is called 4:2:0. Instead of encoding as said before, it is reduced the number of chroma samples (U and V) in both the horizontal and vertical dimensions by a factor of 2, that means each 4 luma samples, 2 chroma samples are encoded(one of each type). We will use this subsampling in the present project, we can see it graphically in Figure 1.2:

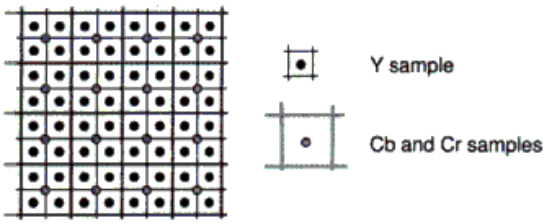


Figure 1.2: 4:2:0 subsampling.

As said in [11], the main goals of this standardization effort are to develop a simple and straightforward video coding design, with enhanced compression performance, and to provide a "network-friendly" video representation which addresses "conversational" (video telephony) and "non-conversational" (storage, broadcast or streaming) applications.

The scope of the standardization is shown in Figure 1.3, where we can observe the typical video encoding/decoding scheme. As in prior similar standards, only the central decoder is standardized (imposing conditions in the bitstream and syntax). Therefore several changes in the encoder can be done in order to optimize some characteristics in several applications (inducing a higher flexibility), always taking into consideration that the decoder has to be able to understand the incoming bitstream obtaining as a result the same video than in the encoder if there have not been errors in the data transport.

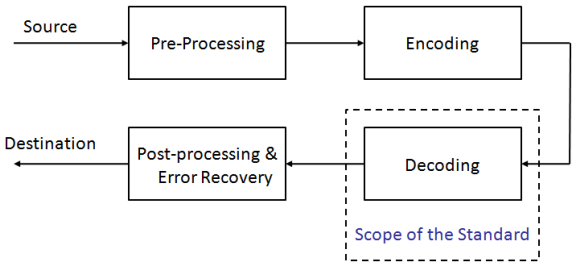


Figure 1.3: Scope of video coding standardization.

The H.264/AVC design comprises a Video Coding Layer (VCL), charged of efficiently represent the video content, and a Network Abstraction Layer (NAL), which has the objective of arranging the VCL representation of the video and pro-

viding proper headers in order to conform the bitstream according to the transport layer. We can see the scheme in Figure 1.4:

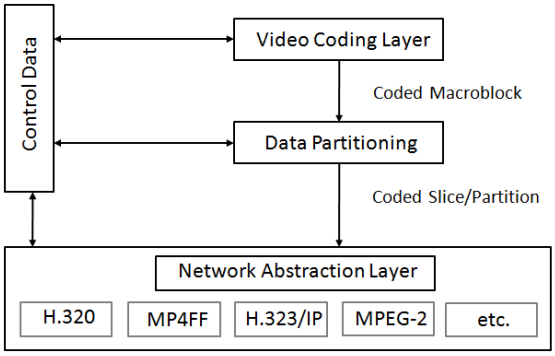


Figure 1.4: Structure of H.264/AVC video encoder.

In the next sections we are going to see the structure of the NAL briefly, the main characteristics of VCL (in details because the most relevant part in our project), the profiles and the software used.

### 1.2.2 Network Abstraction Layer (NAL)

The NAL is designed to provide "network friendliness" to adapt in a simple and effective way the use of VCL for a huge variety of systems. The NAL maps H.264/AVC data to transport layers as RTP/IP, H.32X or MPEG-2.

The smallest unit of NAL is the NAL unit, consisting of a packet containing an integer number of bytes. NAL units have proper headers concerning information contained in its payload. There are different NAL units, VCL and Non-VCL NAL units. VCL NAL units contain the data representing the values of the samples in the video pictures, and Non-VCL contain additional information as parameter sets, that can change the decoding of the bitstream sent in the VCL NAL units.

A set of NAL units in a specified way is an access unit (its decoding results in one decoded picture or frame), and a set of access units compose a coded video sequence. The access units are sequencial in the bitstream. Within the bitstream the NAL can vary the composition of the parameters in order to improve and make more efficient the transport and specially the error detection.

### 1.2.3 Video Coding Layer (VCL)

We are not going to go into details here in technical characteristics of the standard, but a general overview of the working of the most important part of the standard will be explained, for more detailed information, [12] can be seen.

The VCL consists of a hybrid of temporal and spatial prediction, in conjunction with transform coding. Each picture is partitioned in rectangular blocks of 16x16 pixels called macroblocks (and denoted as MB), and these macroblocks are encoded with the associated luma and chroma samples. All these samples will be either spatially or temporally predicted, and the resulting prediction residual will be sent after the application of transform coding and quantization.

The macroblocks within a picture are grouped in slices, a picture maybe split into one or several slices. Each slice is encoded and decoded independently, that means that a slice does not use the data of other slices within the picture to be decoded and reconstructed, and is racket separately. Normally, the macroblocks are grouped in slices in raster scan order as shown in Figure 1.5 left side, but multiple options of Flexible Macroblock Ordering (FMO), a slicing strategy, can be used to divide a picture into slices. An example of FMO can be seen in Figure 1.5 right side. At this point we only have to know that macroblocks' encoding does not have to be in a fixed way, but we can exploit this characteristic to send particular macroblocks with different parameters. We will focus in the FMO in the section 1.3.

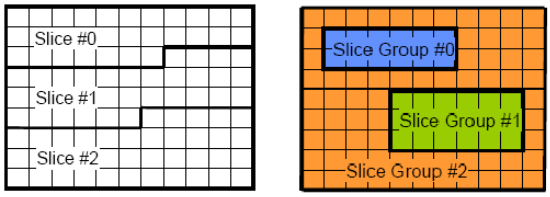


Figure 1.5: Slicing Strategies.

Each picture in the H.264/AVC standard can be predicted with spatial prediction, temporal prediction or both of them. When Intra prediction is used the

spatial dependencies are exploited, and when Inter prediction is used the temporal dependencies are exploited. We will explain briefly both cases:

- **Intra prediction (spatial prediction),** each Block in an Intra frame is predicted using spatially neighbouring samples of previous encoded blocks in the present picture only. The encoder chooses in this case the best MB to make the prediction, and this prediction is sent together with the residual. Images encoded with intra prediction require much more bits to be transmitted than the ones encoded with inter prediction, but the quality is higher. Obviously the first image of a sequence has to be encoded with intra prediction.

- **Inter prediction (temporal prediction),** this prediction exploits the temporal statistical dependencies. Each sample in an Inter frame is predicted using previous encoded frames without using data from the present picture. The encoding process (motion estimation) in this case consists of choosing motion data, comprising the reference picture, and a spatial displacement that will be applied to the samples corresponding to the MB. In the chapter 2 we will focus more deeply in this topic.

The residual of the prediction (Intra or Inter), being the difference between the real and predicted block will be sent together with the corresponding additional information. Within a frame encoded with Inter prediction, some macroblocks with Intra prediction can be encoded. Within a MB, more sub-partitions can be done, the more partitions there are, the more information has to be transmitted.

This standard supports five different coding types: I and SI (intra prediction) and P, B and SP (inter prediction). The simplest ones are P and I (corresponding to the baseline profile), B is used for biprediction (a picture referred to two previous pictures instead of one only) and SI and SP work as I and P frames but specified for efficient switching bitstreams encoded at various bit-rates. In this project we will work only with the baseline profile since it is the only profile reccomended by

3GPP, that means that we will work only with I and P frames, and we will exploit the characteristics of P frames only.

We can see the basic coding structure of H.264/AVC for a MB in the Figure 1.6, we can observe that within the encoder there is an implemented decoder in order to conduct prediction for the next blocks of pixels of the next picture having the same reference both at the encoder as well as at the decoder:

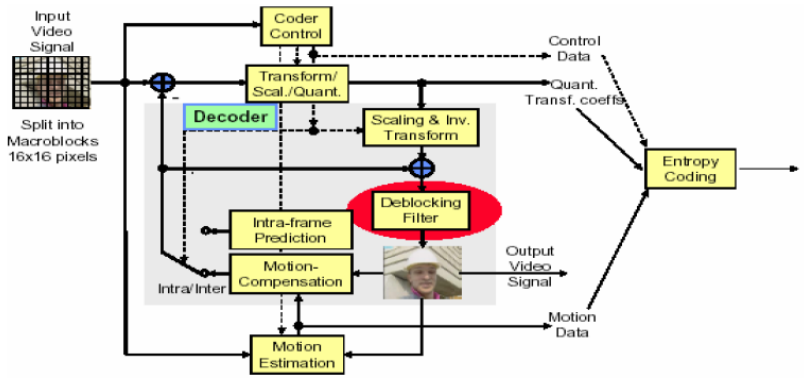


Figure 1.6: Basic structure of H.264 for a macroblock.

After having transformed (with a separable integer transform with basically the same properties as Discrete Cosine Transform(DCT)) and quantified the macroblocks' samples in the specified order, two different methods of entropy coding can be used: a method called Context-Adaptive Variable Length Coding (CAVLC) and a sophisticated method called Context-Adaptive Binary Arithmetic Coding (CABAC). In our case the simplest option (since we are working with the baseline profile), CAVLC, will be used.

### 1.2.4 Profiles and reference software

Due to the quantity of applications covered by the standard, a few profiles have been designed to avoid the implementation of all possible stream structures, in that way it is made easier the interoperability between similar applications. A profile is a subset of features, it defines a set of coding tools that can be used to conform a particular bitstream. In H.264/AVC three profiles were defined at

the beginning comprising all the possibilities: the Baseline, Main and Extended Profile. Nowadays the range of profiles have been extended to seven, but they will not be commented. These are the main characteristics of the three main profiles of the standard:

- **Baseline Profile:** designed for low complexity and low rate applications, widely used in video conferencing and mobile applications, it is the primest of all profiles, containing the common parts of all profiles: I slices, P slices and CAVLC, and some particular features like Flexible Macroblock Ordering (FMO), that will be explained in the next chapter.
- **Main Profile:** designed for Digital Storage Media and TV Broadcasting, it supports B slices, weighted prediction, CABAC, field coding and MB adaptive switching between frame and field coding.
- **Extended Profile:** designed for more error prone environments due to its robustness and server stream switching, including SP and SI slices.

In this project we will use the baseline profile since we are working with a mobile low rate application.

#### Reference software

In this thesis we are working with the H.264/AVC encoder reference software developed by JVT for testing, the Joint Model reference software(JM) [13] version 12.2 (FRExt). We are modifying this encoder's software in order to achieve the objectives that will be described in the next chapter. We will exploit the possibilities of the standard and we will explain the modifications made in the functions along the thesis (deeply in the chapter 3).

## 1.3 Previous work

This present project have been made in the "Institut für Nachrichtentechnik und Hochfrequenztechnik" in the Vienna University of technology, with the supervision

of Luca Superiori and professor Markus Rupp. I have carried on with the main objective of previous works completed, the encoding optimization of soccer video sequences, specially with the one seen at [14], that will be overviewed now. It is exploited the subjective importance of the regions composing the soccer videos, in order to, taking into consideration the perceived quality by the user, apply and exploit some characteristics to reduce the quantity of bits transmitted. In this chapter we are going to see the previous work made, which has caused the possibility of researching the topic of this project.

Due to the known problems of screen size (small resolution), blurring effect due to reduced frame rate and need of high compression, the outcoming result of soccer video encoding is an unsatisfactory quality percieved by the users in a mobile phone video streaming scenario. Because of that it becomes necessary the research of new methods and possibilities to improve that. An example can be seen in Figure 1.7, it can be observed the original video and the resulting video after being encoded with a reduced bit rate with H.264/AVC standard without the use of any kind of slicing strategy:

Figure 1.7: Original and encoded frame with reduced bit rate.

Because no kind of slicing strategy has been used, the bits containing the ball, the players and the other information are delivered in the same packets. That means all the samples (or macroblocks) are encoded with the same quality (or with the same quantization parameter (QP), explained later). Observing the encoded video, the objects cannot be discerned very well (specially the ball), but the field can be perfectly recognized. It can be supposed that a lot of bits are necessary to encode properly the ball because it is a very small object moving all the time in

random directions, whereas the field does not change a lot from one picture to the next one.

The main idea of the investigation is the subjective importance of the regions of video soccer frames. It can be imagined which regions are really important from the observer point of view, the most important parts are the ball, the players and the lines, whereas the field (only has to not to be affected by blockyness) and the audience (is not relevant for the understanding of the match) will not be so important. These characteristics can be exploited to transmit the most important parts using more data rate and the least important parts with less data rate. Making use of the possibilities in the standard to perform data partitioning, Flexible Macroblock Ordering (FMO), commented previously, can be used to encode different group (slices) of macroblocks with different characteristics. The target of this investigation is to obtain a segmentation at macroblock level.

One of the characteristics of the new H.264/AVC Standard is the possibility of splitting a picture in regions called slices. Each slice contains macroblocks encoded sequentially in raster scan order (from left to right and from up to down). Each slice can be decoded always independently. FMO consists of deciding to which slice each MB in the picture corresponds. Each MB is assigned to a slice according to a MBAmap (macroblocks' situation map) consisting of an identification number for each MB in the picture specifying the slice. The number of the slices is limited to 8. Without FMO, images consist of one only slice with the encoded macroblocks in scan order and the same encoding properties. The use of FMO allows error recovering exploiting the spatial redundance of the pictures, specially using type 1 in Figure 1.8. If one slice is lost in the transmission, is easer to reconstruct the neighbouring blocks, due to that the use of FMO is proper for error prone environments.

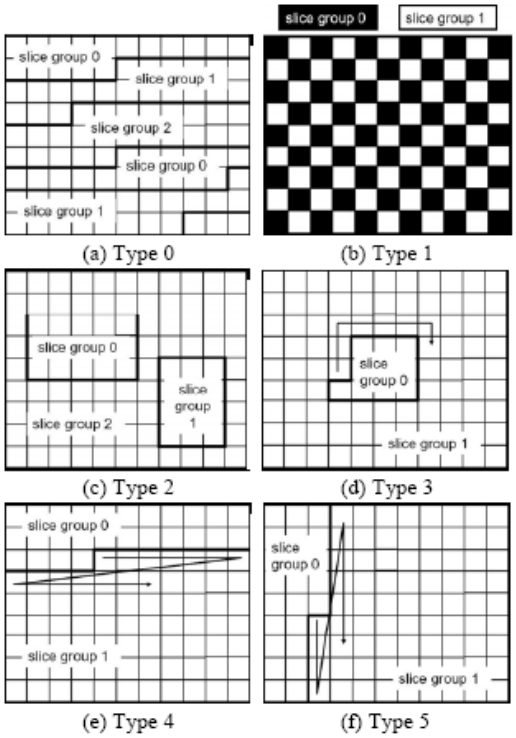


Figure 1.8: Different types of FMO.

The MBAmap indicates the slice corresponding to each MB, this map is already known at the decoder without the necessity of sending it because there are some patterns already created. Only some short parameters have to be sent to indicate the number of macroblocks contained in each slice, excepting in the seventh type. FMO has seven different types, from type 0 to type 6, the first six types (from 0 to 5) contain a certain pattern as can be seen in the previous Figure 1.8:

The last type has to be input to the encoder by means of a file, and in this case the MBAmap will have to be transmitted entirely. This case is the most flexible, but has the disadvantage of sending a map for each picture. This, in conjunction with the additional data in the headers, increase the data to be transmitted, but improves the objective and subjective quality (SNR o MSE) as said before and allows to encode macroblocks with different quality within the same picture. This explicit type of FMO is used in this previous project to encode the different zones of a football frame in a different way (and also will be used in this project), for

more information about FMO, [15] can be seen.

To obtain the maps that are going to be input explicitly to the encoder, the video has to be preanalysed with MATLAB to segment the video in three zones at macroblock level. It was thought to segment each frame in 3 zones: the field (zone 0), the players, ball and lines (zone 1) and the audience (zone 2). Applying recognizing methods like ball detection and motion vectors distribution each frame is partitioned into three zones as can be seen in Figure 1.9:

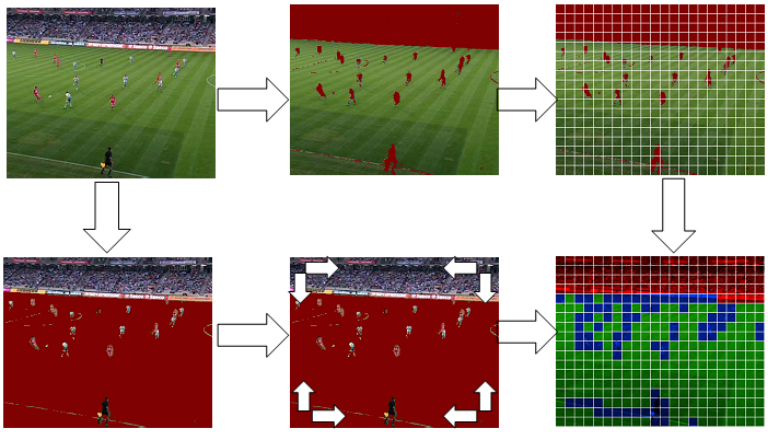

Figure 1.9: Frame segmentation in three zones with Matlab.

The explicit allocation map has to be input to the encoder in order to use the FMO type 6, in such a way that the encoder is able to encode the different slices with different characteristics. A file indicating the position of each MB is input to the encoder for each picture, it can be seen graphically in the Figure 1.10. After performing the data partitioning, it is necessary to adapt the "distribution" of the bits to the relative relevance of each region. They can be ordered for importance: from the subjective quality point of view the most important zone is the zone 1 obviously, and the other two are less important.

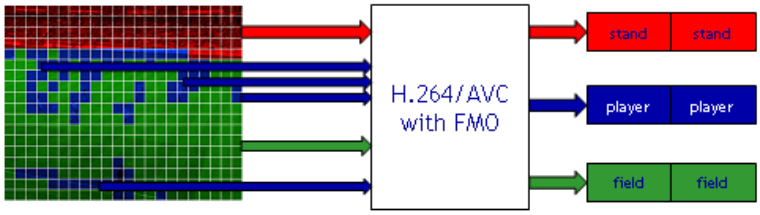


Figure 1.10: Reordering macrolocks into slices.

After inputing the maps to the encoder, it is necessary to optimize the encoding and find the optimal rate-distortion model for the different regions. Each slice is transmitted independently in packets, containing information in the headers about the decoding (it has to be considered that the pictures used as reference are equal in the encoder and in the decoder).

A standard decoder uses a single QP for all the MBs; without breaking the standard, the simplest way to decrease the bit rate for a slice is varying its Quantization Parameter (QP), the QP can be modified in order to compress more the least important zones of the video frames. The QP is used for determining the quantization of transform coefficients in the encoder, the quantized transformed coefficients are scanned in a zig-zag fashion and transmitted using entropy coding methods. The QP can take 52 values (from 0 to 51), when QP increases, more compression is applied (many HF components are lost and a worse quality at the decoder side is measured). The three slices containing three different zones can be encoded with different QP, in that way less bits will be used to encode the least important parts. To encode with different QPs the three slices, the standard encoder was modified to vary the QP according to the slice being encoded.

To encode the videos, the first frame was encoded as I frame with a low QP (because the reference frame should contain as much detail as possible) and the following as P frames. The results obtained have shown a good improvement in terms of bit rate saved and a better perceived quality using the same bit rate.

## 1.4 Main problem and proposed solution

In this subsection we are going to explain the origin of the thesis, and later we will try to explain the proposed solution without going into very technical details, because it will be explained along the memory.

### 1.4.1 Main problem

As we have seen in the previous subsection, now we dispose of the possibility of encoding the macroblocks within a picture in different ways, according to the slice they belonging to. Soccer frames can be partitioned into three zones and encoded with different qualities (QP), the example that we are going to visualize was encoded with the standard encoder (that means without using the previous method) with a QP equal to 26. We are going to see a graph indicating the amount of information sent for each MB in a frame, we see the Figure 1.11. In that way we will know how much each region affect the coding efficiency.

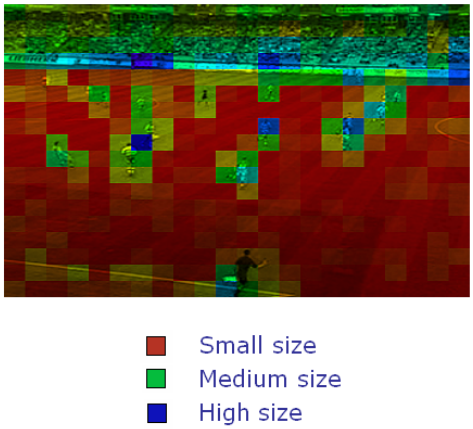


Figure 1.11: Amount of bits sent for each MB.

In this picture we can see the size associated to each encoded MB, red colour means small size, green means medium size and blue means big size. We can observe how the macroblocks of the audience are encoded with more bits than the most of the others. Being this region not so important from a subjective

point of view, it results to be suboptimal in terms of making the best use of bandwidth. It can be said the soccer encoding is not too efficient from this point of view. Partitioning (with Matlab) the frame as seen before into three different zones with the previous work, we obtain the allocation map. If we observe the resulting allocation map, compared with the previous Figure 1.11, we can observe how macroblocks corresponding to the audience contain more bits, we see the comparison in the Figure 1.12:

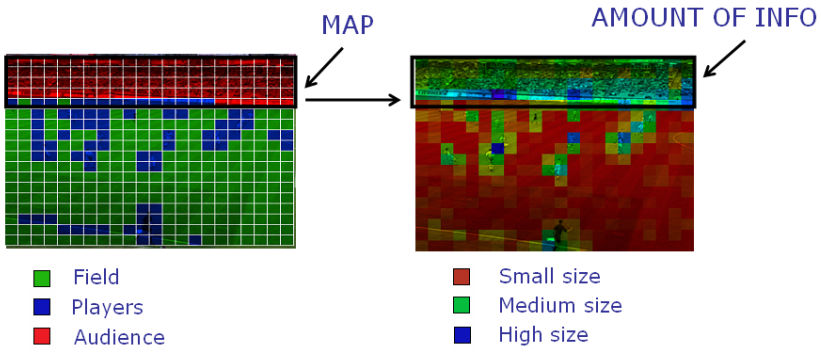


Figure 1.12: Allocation map vs amount of bits sent.

We can put the question: "Why are the MB associated to the audience so big?". The answer is because these macroblocks contain high frequence patterns and they are quite hard predictable from previous frames (the temporal prediction doesn't work efficiently). The encoder exploits the fact that videos contain much more information in low frequencies, and because of that high frequencies are more compressed. If the QP increases more compression is applied and therefore more information is bypassed.

The video frame has been divided into three regions, that have different characteristics, we are going to analyze them:

- The field only contains low frequency, so it's very easy to predict.
- The players are very hard to predict, because their movements are unpredictable. Because of that so many bits are necessary to encode them.

- The audience can be considered "static background".

The main problem is the wasting of bandwidth in a considerable way in soccer encoding, specially in the parts where audience is encoded. Taking into consideration that the audience can be contemplated as the least important part of the soccer encoding, we can think about improving or modifying its codification.

### 1.4.2 Proposed solution

This project continues with the main objective of reducing the bit rate used to transmit soccer video sequences, we want now to optimize the encoding. Because more bits (respecting the other regions) are used to transmit the audience and it can be considered as the least important part in a soccer video, we are going to propose a possibility to encode it differently.

At this point it is important to comment an important concept that will be explained later deeply, but a very short idea about it has to be known to understand the proposed solution. We talk about the Motion Vector (denoted as MV). The prediction signal for each macroblock is obtained by displacing an area of the corresponding reference picture, which is specified by a translational motion vector and a picture reference index.

To make easier the understanding, we show a graphical example in Figure 1.13, we imagine we are encoding a MB in the frame N (P frame), we encode the MB containing the beginning of the pink advertisement (rounded with a black circle). The encoder finds the best MB to predict in the previous frame N-1, and then it calculates the MV (in green). This MV will be sent together with information about the encoding of the MB and the residual (difference between the real and predicted MB). For each encoded MB, the MV, some encoding information and the residual are transmitted.

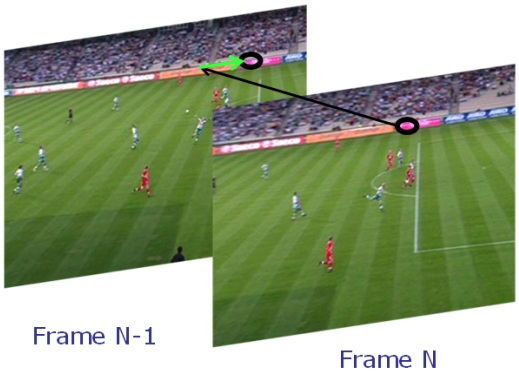


Figure 1.13: MV concept, encoding a MB.

An important thing to consider at this point is the information that can be obtained with the analysis of the MVs. In the previous example, the MV was pointing to the right, and we know that camera is moving to the right. This behaviour is due to the characteristic of the audience of depending on the movement of the camera, because people at the audience are not going to move, and camera movement does not depend on them. Because of that, the value of the MV of the MBs at the audience can give us a idea of the movement of the play due to its "static background" characteristic, thing that we are going to exploit, as we are going to see later.

We are going to see the motion vectors distribution for the audience in the whole sequence. In the Figure 1.14 we can see that the distribution presents a visible peak and that there are not tails:

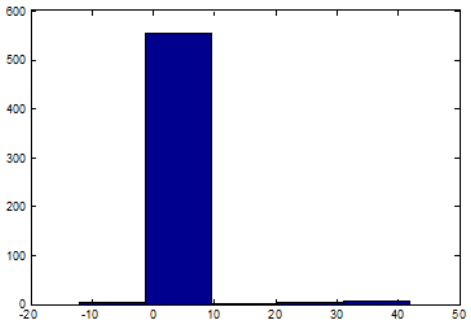


Figure 1.14: Motion vectors' audience histogram.

This fact demonstrates that motion vectors are concentred in a small region, that means that most of them point in the same direction. Therefore we can predict the camera movement observing the average of the motion vectors associated to the audience and exploit the camera movement to predict a "Global Motion Vector" (denoted as GMV).

Since the audience can be considered "static background", our proposal is to exploit the property of the audience to be static. We can think about sending only one motion vector for the whole audience indicating the camera movement to copy macroblocks directly from the previous image, but shifted. H.264/AVC supports multi-picture motion-compensated prediction. That means, more than one previous picture can be used to predict the mv, but as we are thinking about using the information of the previous picture, we will use only the previous one.

We can see a simulated example of the idea we want to apply in the Figure 1.15. We have used for this example two encoded consecutive frames, and the part marked with a black rectangle would be the part copied and shifted in our future encoding method.

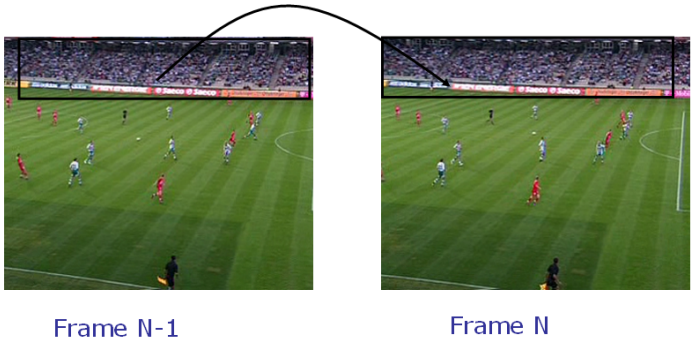


Figure 1.15: Example of the proposed method.

To sum up, we realize the next tasks:

- Implement in standard H.264 codec (JM) a global motion compensation for the whole slice containing the audience.
- Modify functions and normal working of the encoding, paying attention to

the new macroblocks appearing at the border (problem that will be explained later in the chapter 3), in order to create of a new encoding method for the audience slice.

- Implement a "zoom detector" in Matlab, the proposed method does not work if there is zoom. When zoom happens, the size of the image associated to each object changes, and it is not possible to represent a image with the information of the previous one. It will be widely explained in chapter 4, dedicated to this topic.

- Investigate the possibility of extracting the global motion vector from the field.

- Investigate the rate-distortion of the proposed method, taking into consideration that we know the PSNR (objective quality) will decrease considerably due to we are not going to transmit residual, but keeping the trust of the subjective (perceived) quality will not be bad.

## 1.5 Outline of the thesis

The aim of this thesis is the research of a new possible encoding strategy for the audience in soccer video sequences. We work with with the H.264/AVC in baseline profile, in a video streaming context over UMTS networks, but in a simulated scenario because the videos have to be preanalysed with matlab. We work mainly with P frames using the special slicing strategy FMO type 6 (explicit map).

Throughout this diploma thesis, we are going to explain the steps done in the thesis in chronologic order. We summarize the contents of the chapters in this subsection. The present thesis is organized as follows:

- In Chapter 2, we will study the specific encoding of a P frame (paying special attention to the Skip mode selection) and its decoding in order to try to find the manner to use a GMV for the whole audience as said before.

Afterwards we are going to implement the new encoding method modifying some functions of the software given in [13], in Visual C++.

- In Chapter 3, we are going to preanalyze the soccer videos with matlab in order to obtain information about the videos. We will extract the movement of the camera, that will be used in the encoder to shift the audience each frame. The camera movement will be also used to address the borders problem (it will be presented and solved later).

- In Chapter 4, we are facing one of the main obstacle of the encoding method: the zoom. When there is zoom the method cannot work properly, because the size of the macroblocks changes and makes it impossible (not impossible but yes very annoying) copying directly information from the previous picture. Some state of the art of zoom detection will be presented. Afterwards the zoom detector made for this project is presented.

- In Chapter 5, we analyze the results obtained after the encoding (after having analyzed the videos to detect zoom and provided proper information to the encoder). We compare objectively and subjectively (by means of a subjective quality test) the encoding results in terms of bit rate and quality obtained. We will have to pay attention to the perceived quality of the videos, because as we have commented before, we know the objective quality is going to decrease.

- In Chapter 6, we will discuss the conclusions, as well as the limitations of the project, its scope and some future steps to do.

# Chapter 2

# Video encoding

In this chapter we are going to see the encoding process of a frame in the H.264/AVC standard, in order to investigate the possibility of modifying the encoder to achieve our objective, that is using only one Gmv for all the macroblocks containing the audience for each frame. We will pay special attention to the encoding inter frame mode called skip, used mainly in videoconference applications because we will exploit it. From now, we are working with progressive video, and P(inter prediction) frames within the baseline profile, therefore, when we will have to comment something of the functions of the software, we can bypass the functions useless regarding to this project.

We are going to see the different levels passed through by a frame to be encoded. We are focusing in the MB level (the smallest one) explaining the amount of different characteristics that can be used to encode a MB within a P frame, focusing as said before in the Skip mode. After that, a main scheme of the encoding at MB level will be shown. To study the variables in the JM software that have to be modified, and when, how and where have to be modified. It will be tried to not to show explicit code (unless it was imperative) in this chapter because it would be very annoying for the reader. Instead of that, concepts will be explained by means of schemes and diagrams visually, together with explanations of course.

## 2.1 Encoding process

The H.264/AVC standard contains a lot of possibilities to encode differently the video. Mainly the principal parameters and characteristics have to be set by means of a text file at the beginning of the video encoding. Very different things are set like the percentage of intra frames used per inter frames, the profile and complexity used to encode, the availability of special features, compression level (QP), etc. These parameters affect the encoding at different encoding levels, for example the QP affects the level compression of all the images, and another parameters affect the encoding of a MB within a P frame. So we encode within a hierarchical order in different levels, image level, slice level and finally MB level.

While encoding, information about the codification has to be transmitted in order to indicate the decoder the way to decode properly. This information is contained at the three levels, some information is directly explicit headers and other is self-contained in the encoding process. At image level some headers called Sequence Parameter Set (SPS) and Picture Parameter Set (PPS), are sent containing information about the characteristics enabled and used in the present encoding. SPS are set for a long sequence of pictures, and PPS are set for individual pictures, they are sent in separated packets. Some special characteristics like FMO (explicit map for each picture in case of explicit mode), explained in the last chapter, are set in the PPS. Also headers called Slice Header (SH), are transmitted in order to indicate information about the slice. Of course the information sent in the PPS, SPS and SH affect the macroblocks encoding. We are focusing in this project and specially in this chapter into the MB level, the idea is studying the encoding process to modify its working. We will focus later in the encoding of the slice number 2 (corresponding to the audience according to section 1.3) to modify the code, we are not going to modify parameters in the other two levels. We can see a general scheme of the different encoding levels commented before in the Figure 2.1, we can see our objective is modifying the part marked with a discontinuous line:

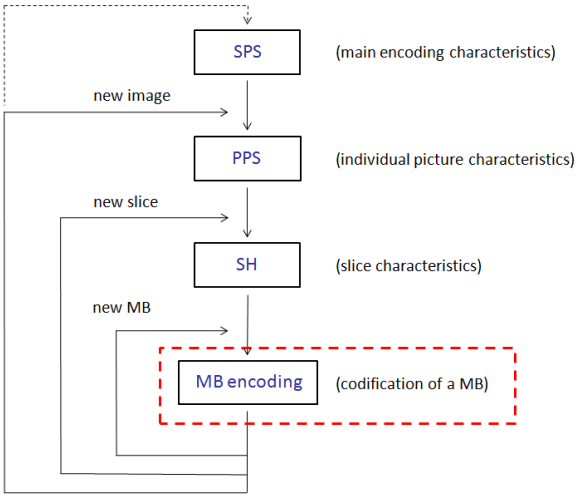


Figure 2.1: Different encoding levels.

Now we are going to explain how a MB is encoded within a P frame and the information that has to be transmitted, without paying attention to I frames. Generally the first picture in a sequence is encoded as I frame because does not exist any previous picture obviously, also in this case the first picture will be encoded in that way, but no encoding modifications will be made.

### 2.1.1 Encoding at MB level.

The VCL design follows the so-called block-based hybrid video approach, because of that, after subdividing the image into 16x16 pixel blocks, the encoding is made at MB level. At this level, each MB with its associated luma and chroma components are encoded. There are several coding types supported by the standard (each of them corresponding to a particular partition of the MB). A MB can be partitioned into smaller sub-blocks, until a maximum of 16 4x4 sub-blocks for MB. According to the slice type and the parameters set in the SPS, PPS and SH, some coding types are available to be selected and used. In our case, two Intra coding types are allowed, exactly 16x16 Intra type and 4x4 Intra type, and all the Inter coding types. We can see the possible Inter coding types (and consequently possible segmentations) in Figure 2.2, each MB can be split into four 8x8 sub-blocks, and

then each of them can be also partitioned into four 4x4 sub-blocks.

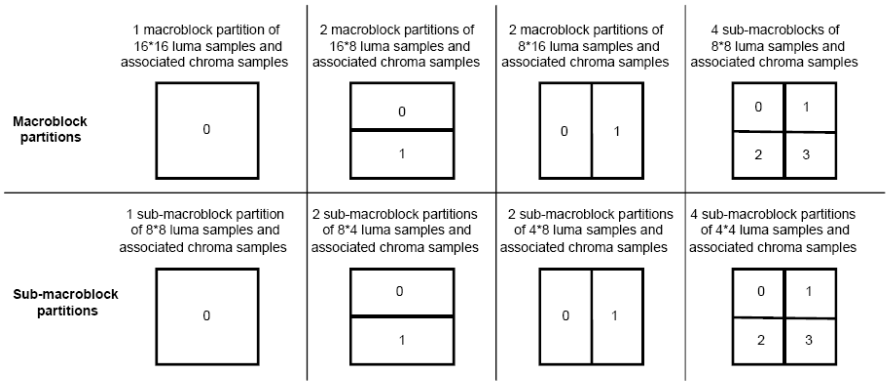


Figure 2.2: MB partitions for motion estimation and compensation.

Focusing in the inter prediction modes, each predictive-coded MxN block (4x4, 8x8 or 16x16 according to the encoding decision) is obtained by displacing an area of the corresponding reference picture specified by a motion vector (explained before), which together with the residual, will be sent. As a MB can be partitioned into 16 4x4 sub-blocks, a maximum of sixteen motion vectors (explained previously) may be transmitted for a single P MB.

When encoding, several options (types) are available to encode. As it can be supposed, the more information has to be sent, more difficult is to predict properly the MB, and more partitions have to be done. If a MB is partitioned into 16 4x4 sub-blocks, more bits are transmitted. To summarize, this information has to be transmitted for each MB:

- MB coding type selected.
- From 0 to 16 (or 32 if we consider the two horizontal and vertical components), motion vectors residuals (differences between real and predicted mv) depending on the selected encoding type.
- Luma and chroma residual for each sub-block, the amount of information here depends on the number of sub-blocks and on the QP with is encoded

the MB.

We can see an example of a encoded MB quantified with QP equal to 26 in the Figure 2.3, we can see a trace file giving information of the encoding, the real data sent are the bits at the right side of the figure, and the information on the left side is for complaining what it is being sent. In this case, as we can see at the right side of the figure, it has been selected partitioning the 16x16 MB into 4 8x8, and after that, the first 2 8x8 sub-blocks have been partitioned into 4 4x4 sub-blocks, the third one into 2 8x4 sub-blocks and the last 8x8 has not been partitioned. Therefore, 11 motion vectors (horizontal and vertical) have been predicted and transmitted with its corresponding luma and chroma residual, the amount of residual sent depends on the QP. We do not show all the residual because it is very long.

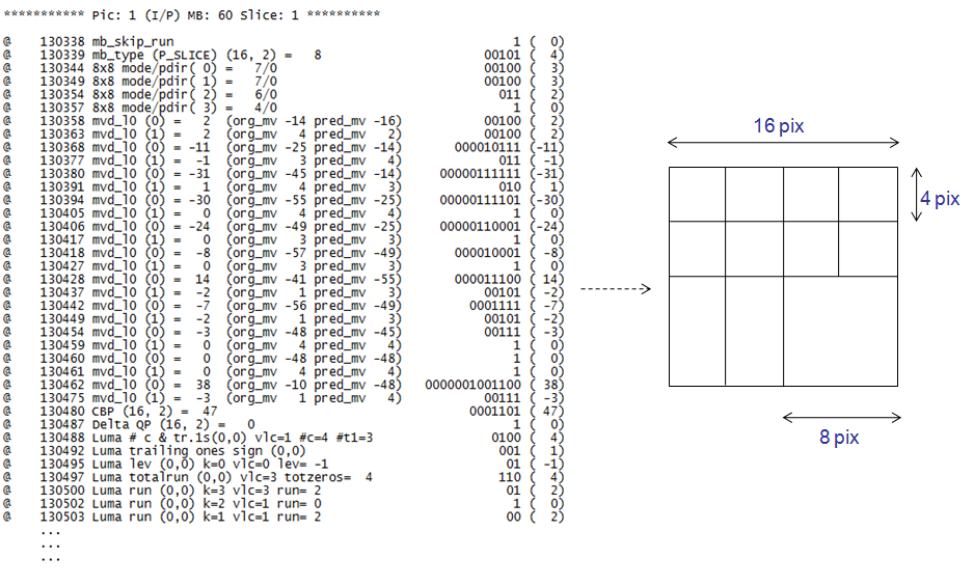


Figure 2.3: Part of a MB bitstream and its resulting segmentation.

In addition to the Inter modes described already, there is one more possibility to encode a P MB, the called Skip mode. The skip mode, as itself suggests, means not sending any information. This type was created for situations where no change or constant motion vector happens. In those cases, a very few bits it would be

necessary, but with the skip mode, even less. For this coding type, neither mv residual, nor luma and chroma residual are transmitted. Instead of sending coding type equal to Inter 16x16, mv residual equal to 0 and luma and chroma residual equal to 0, that would mean a lot of bits, nothing is transmitted. In that way, the signal is reconstructed using as motion vector the predicted mv in the decoder (equal to the mv predicted in the encoder).

### 2.1.2 Exploiting Skip MB mode, the idea.

As we already know, the objective of this project is reducing the bit rate for the audience exploiting the property of the audience to be static (or unless depending on the camera movement). We think about copying (and shifting) the macroblocks containing the audience directly from the previous image but without sending any information, and detecting the camera movement to predict and transmit only one GMV that will be used for all the macroblocks. At this point we can think about exploiting the Skip mode, this coding type allows the decoder reconstruct the MB without the necessity of sending information.

The main idea is sending one GMV once in the first MB (without skip mode obviously) forcing the MB to be 16x16, the GMV to be the one we want and without residual, and forcing the other macroblocks of the audience to be skip, but at the same time, reconstructing them at the decoder with that GMV (taking into consideration that the decoder is the standard one). To do that we will have to find some manner to make that the predicted MVs in the Skip MBs are equal to the GMV. We can see in the Figure 2.4 the scheme of the data we want to transmit for the audience compared with the standard encoding.

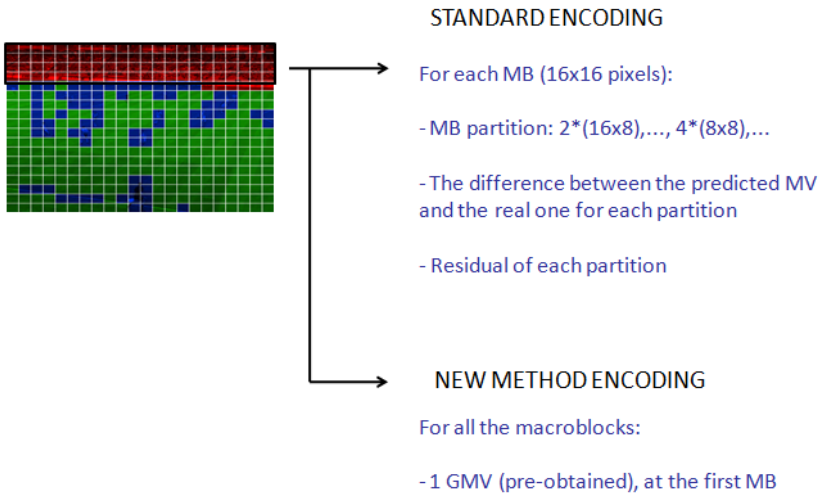


Figure 2.4: Data transmitted.

Obviously we will check later if the system can work, because we have to create some technic or trick to make the encoder and decoder reconstruct the skip macroblocks as we want (with the GMV). At this point, we have to think about how to implement the new possible encoding system. Because of that, the first thing it is necessary to do is diving deeply into the encoding of the P macroblocks in order to be able to implement the solution we are proposing. We will have to study where, how and when are evaluated and set the variables and functions that are charged of the encoding in order to modify the encoding properly.

## 2.2 Modifying the encoder.

We are going to modify the H.264/AVC encoder reference software developed by JVT for testing, the Joint Model reference software(JM)[13] version 12.2 (FRExt), implemented in C++. We will use Visual C++ program to work with.

An important thing to consider at this point is that the decoder is already standardized, because of that all the modifications that we are going to do in the encoding process in the JM reference software have to generate a H.264/AVC

file with a correct syntax and perfectly understandable and decodificable by the decoder at the receiver side. The modifications that will be shown in this chapter are the result of a very hard work, although they are going to be explained in the easiest possible way.

In this project some lines of the code have been modified at different levels and with different objectives. In this chapter we will comment the modifications made at MB level affecting the encoding. Along this document other modifications will be commented.

As we have to be able to modify some part of the code making it decodificable, before that, we have to take a look to the standard encoding. In this section we are going to see the encoding from the point of view of the encoder, that means that we are going deep into the software in order to understand what variables, pointers and functions are charged of the encoding (specially at MB level), and when, where and how we have to modify the behaviour or working of those things.

### 2.2.1 Introducing the encoder.

The encoding, as said before, is implemented at different levels, there are a lot of functions called in hierarchical order from the main encoding function, called lencod, which is called from the command line. We will use this command line to indicate the encoder the video and the characteristics of the encoding by means of a txt file. The standard H.264/AVC generates a H.264 file (only understandable by a decoder), together with a yuv file (video format) representing the output obtained, and the called trace file indicating the information finally encoded and transmitted in the H.264 file, but with its corresponding labels in order to comprise the information sent for each element (as seen in the **??**). All these files are created step by step during the encoding, and because of that there are several variables and pointers used for all of them, and others specific. We will have to take into consideration that when we will change some function, pointer or variable to modify something, this change has to affect the three files in the same way.

As we want to modify the working of the macroblocks' encoding of a particular slice within the P pictures, we will have to add some conditions in order to make the encoder works in a different way. Because of that, we are going to see in the Figure 2.5, a summary showing some of the main functions passed through until reaching the encoding at MB level (function encode_one_macroblock). We show only the main functions used in our encoding case, that means frame encoding (and not field), low complexity(baseline profile used), etc,. We will comment the corresponding assignments to know where are set the main encoding parameters. In that way we will know where to add some conditions to indicate the encoder how to change the encoding.

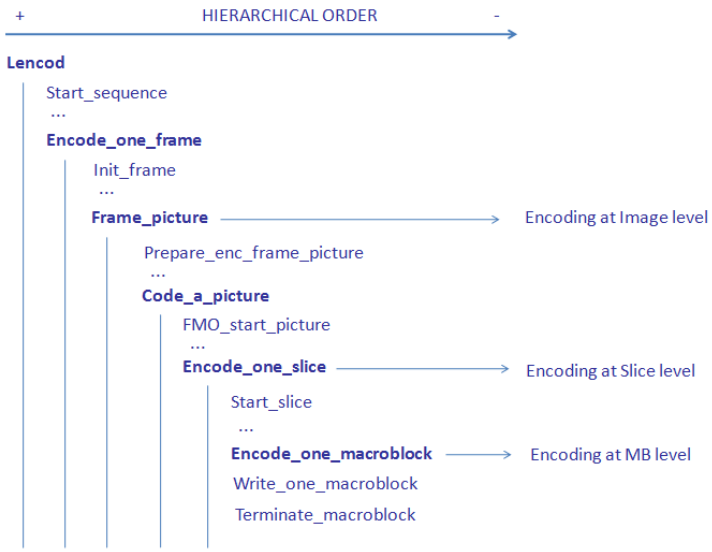


Figure 2.5: Main encoding in hierarchical order.

We will comment the assignments of each function marked in darker:

- **lencod,** main function called from the command line. Several functions not commented are called to make the next tasks. Initializing the global variables and pointers after reading the txt file (configfile) given. It generates the SPS and PPS, opening output files. For all the frames, it calculates the frame number, sets the image type and prepares the parameters and pointers to

encode the image. After, it calls the function Encode-one-frame per each frame. And finally, after all the frames have been encoded, it terminates the sequence, closes all the files and finishes the encoding liberating the resources used.

- **Encode-one-frame,** encodes one I or P frame. It sets the pointers to the frame structures and initializes local frame variables, in this function it is distinguished between frame and field encoding, this function calls frame-picture in our case.

- **Frame-picture,** it prepares the pointers to a frame-picture (and not field) and calls code-a-picture.

- **Code-a-picture,** this is the main picture coding loop, it is called after the image elements have been set up. FMO is established before coding the first MB, the allocation map indicates the macroblocks corresponding to each slice, the QP is set, from here is calles the function encode-one-slice, which is charged of encoding one slice.

- **Encode-one-slice,** sets slice headers and parameters, and for all the macroblocks contained in the slice, it initializes, encodes, writes and terminates the macroblocks.

- **Encode-one-macroblock,** main encoding function at MB level, that will be studied in detail later.

Now we know where we could change some global parameter to change the working of the encoder, we can propound some way to implement the new encoding method. We want to change the encoding of a determined part, because of that, we will have to make the encoder knows in which situations the new encoding method has to work.

We are working with the encoder from the command line, that means that we indicate the encoder the starting of the encoding according to an order. This

order is analyzed in the function configure, just at the beginning of the main function lencod. An example of that is: "lencod -f encoder.cfg -q 30 26 30", where encoder.cfg will contain all the information about the encoding, and "-q 30 26 30" is used to indicate the encoder the different QP levels that have to be used for the three zones.

We can modify the code of the function configure to include the possibility of adding some additional parameters to activate or not a global variable, which will be checked by the encoder in order to use or not the new future encoding method (used to encode differently the audience slices within the P frames). Of course we have to define the global variable in the function global also. From now, we will use from the command line "lencod -f encoder.cfg -q 30 26 30 -b", indicating we want to use our method with the parameter "-b". We can see the added code in the line 210 of the function configure where enable_border_map is the global variable that has been created and that will be used to indicate the use of the method:

```
else if (0 == strncmp(av[CLcount],"-b",2))
{
enable_border_map = 1;
CLcount += 1;
}
```

After that, we have to include some code in the functions at image level (in case of P image) and slice level (in case of 2nd slice) to work differently according to the variable already mentioned enable_border_map. We are going to add some code to the function code_a_picture, which is the main picture coding loop. After being read the allocation map indicating the macroblocks corresponding to the three zones, each slice is encoded independently.

Our objective now is indicating the encoder to encode differently in case of the global variable is activated and we are encoding the audience within a P frame. There is a loop in which each slice is encoded by means of the function encode_one_slice. So just before that, we will check if we are encoding the audience within a P frame and the global variable is activated, and in that case we

will activate another new global variable called audience_skip (also defined in the global function), that will be checked at MB level later to make the encoder work differently or not. We can see here the code:

*if ((SliceGroup==2)&&(enable_border_map==1)&&(img->number != 0))*

*audience_skip=1;*

*else*

*audience_skip=0;*

Along the working of the thesis, the mentioned code added were modified several times to include new possibilities and to solve some limitations, so, as we want to explain the steps made little by little, modifications will be shown along the writing of this present thesis. For example, as we have commented already, if there is zoom the method does not work. Now we do not include this possibility in the code because we have not reached this problem yet, it will be seen later. We show the added or modified lines here, but if someone wants to get deeper in the code, more details can be seen in the Appendix.

From now, we have to investigate the possibility of implementing the already mentioned new encoding method, modifying the decisions of the encoder and modes selected in order to send less amount of bits. Until now, we have explained we have the possibility of dividing each soccer frame into three zones according to a incoming map from matlab to encode differently each of them. And we are going to exploit the characteristic of the audience to depend strongly on the camera movement. Because of that dependence we want to exploit the skip mode together with the prediction possibilities of the standard to use that characteristic to not sending almost information using one GMV for the whole audience in each frame. To use a GMV for the audience in each frame, we have to preanalyze the video with matlab making a mean of all the motion vectors.

After having read the txt file indicating the encoding parameters for the video and having set the variables and pointers properly through all the already seen functions to prepare the encoding, we reach the encoding of each MB. And at this point we are only investigating how implement the method, but without using the

real GMV (that will be calculated in the chapter 3 and used later). Because of that we will use fixed GMV simulating we are using the real ones only to implement the method and check if the method works properly. We are going to get into the encoding at MB level, we are going to see the main encoding function at MB level encode_one_macroblock.

### 2.2.2 Implementing the new encoding method.

Now we are going to see the encoding at MB level, really here at this level is where the hybrid basic source-coding algorithm is implemented, making use of temporal and spatial prediction. We are supposing we are modifying the encoding of the macroblocks of the second slice of each picture. We have to see where, when and how are set the encoding decisions to be able to modify them.

Of course all the modifications we are going to make only will be used when the global variable audience_skip is activated, if not, the standard working will be kept. We are going to see the function encode_one_macroblock.

Mainly, this function is charged of analyzing one MB (all MB are evaluated in this function), and after having analyzed all the encoding possibilities supported, it decides which of all the available coding types will be used to encode the MB. After initializing parameters and pointers, all the possibilities are evaluated, and the best option is set according to the cost. After outcoming this function, the H.264 file, trace_file and yuv file are created according to the values set before.

We can see in the Figure 2.6, a scheme showing the main structure in the coding decision at MB level without getting into details and without naming the functions called.

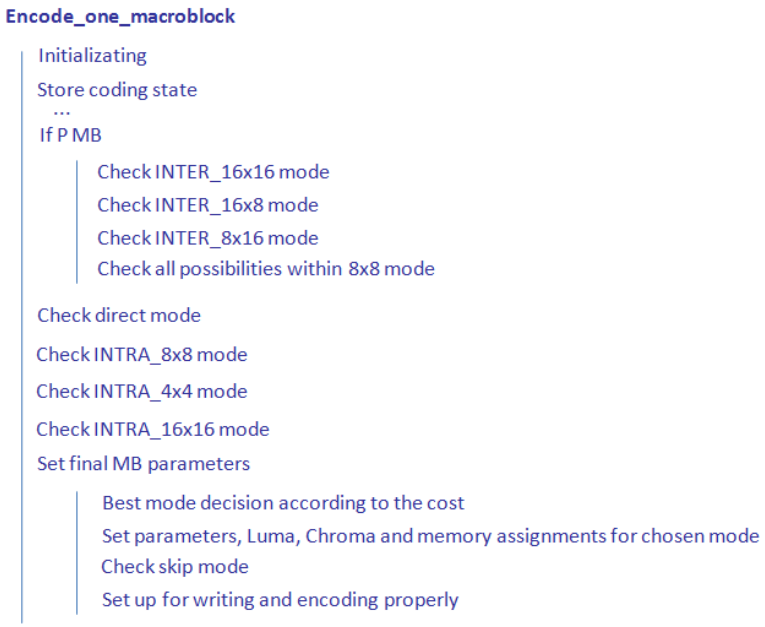


Figure 2.6: Coding function at MB level.

Each MB is prepared to be transmitted in one of several coding types according to the variable enc_mb_valid. As we have said before, there are several possibilities to divide one MB into sub-blocks. These are the 15 possible modes contained in enc_mb_valid, at the right part we can see if in our case the mode is allowed or not:

0 == !Intra (Skip) Allowed

1 == ! Intra & Inter 16x16 Allowed

2 == ! Intra & Inter 16x8 Allowed

3 == ! Intra & Inter 8x16 Allowed

4 == ! Intra & Inter 8x8 Allowed

5 == ! Intra & Inter 8x4 Allowed

6 == ! Intra & Inter 4x8 Allowed

7 == ! Intra & Inter 4x4 Allowed

8 == mode 4, 5, 6 or 7 (general) Allowed

9 == Intra 4x4 Allowed

10 == Intra 16x16 Allowed

11 == IBlock Not allowed

12 == SI_Slice Not allowed

13 == Intra 8x8 Not allowed

14 == IPCM Allowed

The mode 8 includes the modes 4, 5, 6 and 7 because really when mode 8 is selected, each 8x8 sub-block can be partitioned into 4 4x4 sub-blocks, that will be indicated of course.

Our first objective is to force a MB to be skip, coding mode equal to 0. The Skip mode is checked just at the end of the function, this is because really the skip mode is not a coding mode by itself. Really the Skip mode was created to avoid sending unnecessary information with the Inter_16x16 mode, actually these are the conditions that have to be discharged to select the Skip mode automatically by the encoder:

- Inter_16x16 mode selection.
- Not sending residual in the motion vectors, that means the predicted and the real motion vectors are equal.
- The Luma and Chroma residual has to be equal to 0.

In that way, instead of sending a few bits with Inter 16x16 mode, when Skip mode is selected, nothing is transmitted. The first idea that we can imagine to force a MB to be Skip, as the condition is evaluated at the end of the function, is changing the values of the variables at the end to make the encoder choose the Skip mode. We can see here the code analyzed to check Skip mode:

```
//===== check for SKIP mode =====
if ((pslice) && best_mode==1 && currMB->cbp==0 &&
enc_picture->ref_idx[LIST_0][img->block_y][img->block_x] == 0 &&
enc_picture->mv [LIST_0][img->block_y][img->block_x][0] == allmvs[0] &&
 enc_picture->mv [LIST_0][img->block_y][img->block_x][1] == allmvs[1])
{
```

*currMB-mb_type = currMB-b8mode[0] = currMB-b8mode[1] = currMB-b8mode[2] = currMB-b8mode[3] = 0;*

*currMB-luma_transform_size_8x8_flag = 0;*

}

Some conditions are checked to set the Skip mode, the frame has to be a P frame, the Inter_16x16 mode has to be selected (previously), the cbp indicating the amount of residual has to be equal to 0, the image selected to be used as reference has to be the previous one, and the predicted and real MVs have to be equal. After modifying these lines to force these conditions to be true (or simply forcing the mode to be 0), we have achieved to change the trace_file, but not the H.264 file and yuv file in the same way. It could be thought that after the decision, that MB will be encoded consequently, but not. This is because really the parameters charged of the encoding (that will be implemented afterwards) are calculated before, and this is only to set some final parameters. That means that really at the end of the function the Skip mode is checked to write properly only the trace_file, but the values of the variables and pointers of the Skip mode to represent the video are set before, at the checking of the Inter_16x16 mode obviously. Because of that we have to get deeper in the codification of the Inter_16x16 mode to be able to change the code to force the MB to be skip.

We have to make changes to force these conditions to be true, but when they are being calculated. To skip a MB, the best mode chosen has to be the mode 1, because of that we are going to analyze the function step by step observing all the called subfunctions when mode 1 (Inter_16x16 mode) is being evaluated. The first objective is forcing the best mode to be the mode 1. Paying attention to the loop where is evaluated the mode 1, we can observe the next subfunctions in the Figure 2.7:

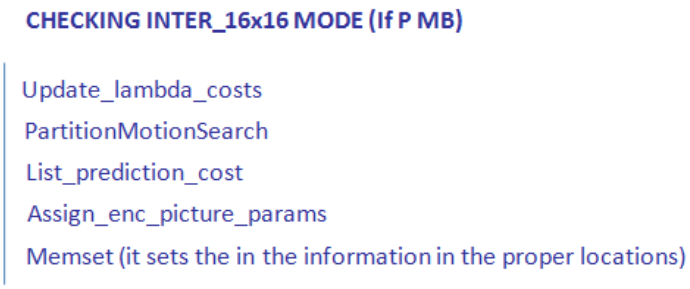


Figure 2.7: Checking 16x16 mode.

The really important function in terms of encoding is the PartitionMotionSearch, which calculates the motion vectors and the Luma and chroma. After the analysis of each mode, the cost given is compared with the previous one, and the smallest one is maintained. So, we add some code to this function to force the given cost to be 0, in that way we force the MB to be encoded as Inter_16x16, even if the encoder had chosen another one. We see the added code to the function encode_one_macroblock just in the line 222:

*if ((mode==1)&&(audience_skip==1))*

{ *cost=0;* }

Before going into the next function, we have to take into consideration that there are some main big data structures like:

*Img,* containing all the information concerning about the image, its current parameters and its current encoding.

*Enc_picture,* containing information about the encoding to create the H.264 file.

*Enc_frame_picture,* containing information about the encoding to represent the yuv file.

All these commented structures are very big and contain very similar information, they have different commitments, and we are working with them. Globally they work together to make possible a so difficult encoding.

Until now, we have forced the MB to be encoded with the mode Inter_16x16.

Our next objective is to know where and how are set the motion vectors, and the variables used, to modify them to force the predicted mv to be equal to the real one, another condition to skip a MB. To do that we have to go into the function PartitionMotionSearch in depth, let's see in the Figure 2.8:

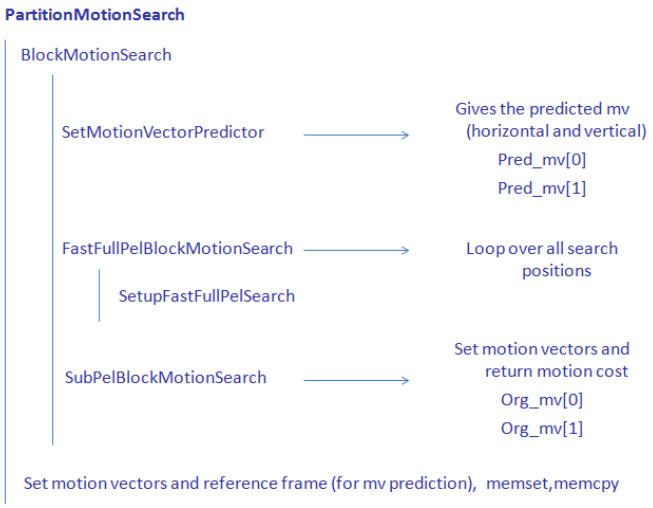


Figure 2.8: PartitionMotionSearch function.

Really are shown only the most relevant functions without going into details. We summarize the concepts shown because otherwise if would be very hard and long to explain the concepts here. Actually the calculation of motion vectors is more confusing and it is obtained continuously along the function PartitionMotionSearch, but the schemes act as summaries to make easier the understanding.

To force the real and predicted motion vectors to be equal we have to modify at the end of the outcoming functions some code lines. We can think about modifying the values of the real and predicted motion vectors to be 0 (we start for the easiest case obviously), that would mean no movement. We add these conditions at the end of the functions:

In SetMotionVectorPredictor, where are calculated the predicted motion vectors:

*if (audience_skip==1)*

*pmv[hv]=0;*

In SubPelBlockMotionSearch, where are calculated the real ones:

*if (audience_skip==1)*

{

**mv_x=0;*

**mv_y=0;*

}

Where mv_x and mv_y represent the real motion vectors associated to the Inter_16x16 mode, and pmv[0]and pmv[1] represent the predicted ones. Those values will be copied and set properly at the end of the function BlockMotionSearch, and later at the end of the function PartitionMotionSearch, reaching to the main encoding function encode_one_macroblock, where it will be set and analyzed. They will be copied from some local variables to another global variables to finish being placed in img, enc_picture and enc_frame_picture.

After changing these lines, we check what we have realized until now. We have got to set the MB coding mode to be Inter16x16 and to force the real and precicted motion vectors to be equal, that means no motion vector residual will be sent. We must have in mind our objective, that is forcing the macroblocks to be skipped, but encoding them with one GMV(equal in all the macroblocks at the audience).

We have researched about MB encoding, specially about Inter_16x16 mode. Before forcing the predicted and real motion vectors to be equal, we have have observed the working of the motion vectors prediction. We have seen that when a MB is encoded as Inter_16x16 MB, its predicted mv is equal to the real mv of the previous MB in case of also has been encoded as Inter_16x16 MB (as is our case). In case of being the first MB, as there is no previous encoded MB, the predicted mv es equal to 0, and the residual sent to the decoder is equal to the real one(residual=real-predicted). The decoder reconstructs the image using the real mv, it predicts the mv in the same way than the encoder, and with the residual, it obtains the real one to reconstruct. Because of that it has no sense to force the prediction in the encoder because the decoder will predict another one.

The best solution we can make is exploiting the characteristic of the mv pre-

diction. As we know that the first MB (encoded as 16x16 of course) will predict the mv as 0 because it does not have reference. We can force only the real mv to be the one we want (the GMV), in that case the residual in the first MB always will be the GMV. In that way, the next predicted motion vector will be the GMV (the real of the first MB), and forcing the real motion vector to be the GMV again, no residual will be sent anymore, but we will decode it with the GMV. We have achieved our objective, forcing the real and predicted motion vectors to be equal (except the first MB), and making a system to encode all the macroblocks with one GMV. The solution then is only forcing the real MVs to be the ones we want.

Now, the MB is forced to be Inter_16x16 with the real mv to be 0, to complete all the conditions to skip the MB, we have to force the residuals not to be sent, that means forcing the luma and chroma residual to be 0. We go back to the main encoding function encode_one_macroblock. After the analysis of all the modes, where we have made the encoder decide to encode all the macroblocks as Inter_16x16 macroblocks, now we reach the setting of the final MB parameters, that means, according to the best mode, prepares all the structures and fields fullfilled to encode consequently after outcoming the function. One of those commitments is setting the Luma and chroma, that will be used to encode the H.264 file, video file and trace_file.

The Luma and Chroma are evaluated in the functions LumaResidualCoding and ChromaResidualCoding. To avoid talking about unnecessary additional code we only say that tese functions are called just before checking the Skip mode condition.

When Luma is encoded, some parameters are transmitted:

- **Coded_block_pattern (only one per MB),** It specifies which of the six 8x8 blocks - luma and chroma - contain non-zero transform coefficient levels. For macroblocks with prediction mode not equal to Intra_16x16, coded_block_pattern is present in the bitstream with the variables CodedBlockPatternLuma and CodedBlockPatternChroma.

- **Luma # c & tr. 1s,** this element shows us the number of the coefficients and the number of the trailing ones different from zero.
- **Luma trailing ones sign,** the signs of the trailing ones are fixed length encoded and do not influence any of the following parameters.
- **Luma totalrun,** this element specifies the total number of zeros before the last non-zero coefficient.

In order not to send residual, we have to set all these parameters equal to 0, the cbp and the non zero coefficients. We have added these lines just after the real calculation of the Luma (line 1430) in the LumaResidualCoding function:

```
if (audience_skip==1)
{
sum_cnt_nonz=0;
currMB->cbp=0;
currMB->cbp_blk=0;
}
```

In that way, no Luma residual is set. We add also this code (line 1915) at the end of the function ChromaResidualCoding:

```
if (audience_skip==1)
img->mb_data[img->current_mb_nr].cbp=0;
```

and this one (line 1834) at the beginning:

```
if (audience_skip==1)
skipped=1;
```

This last modification at the beginning of that function was possible because we searched which variable was charged of containing the values of the Chroma to be encoded when writing the video, and we have investigated where this variable was set, and the search took us to this condition.

The modifications that have been done and shown here are the product of a very hard investigation, because it has been necessary to deal with different subfunctions, data structures containing the data, etc. The modifications made

until this moment in the code are not the final ones, because we explain the project as has been made, in that way, it will be easier to understand. Some little variables will be added later when we use the GMV and another video preanalysis results.

## 2.3 Checking the method.

Since we have reached finally the main objective, skipping the macroblocks at the audience finding some manner to encode all the macroblocks as skip, but reconstructing them at the both sides (encoder and decoder) with one GMV (fixed by the moment by us), that is transmitted in the first MB, but used for the whole audience. This is the moment of analyzing the working of our encoding method with simulated GMVs. The encoder has a H.264 syntax, used to create the H.264 file, but at the same time it outputs a text file showing the meaning of the bitstream and a video file simulating the resulting decoding at the decoder side. We have to check if the H.264 file has been created with a correct syntax (understandable by the decoder) and if the other two files are consequently modified with the modifications that we have made.

We are going to see an example of a video where we use the method forcing the real mv to be zero, it means the audience will not move. We are going to see the trace_file, the video file and the video resulting of the decoder with the H.264 file. We start with the trace_file. In this case, as we fixed the GMV to be equal to zero, and we force the real mv of the first MB to be zero, no residual is sent, we can observe a part of the coded macroblocks at the audience in the Figure 2.9:

```
@    239460 Luma # c & tr.1s(1,3) vlc=0 #c=0 #t1=0                    1 (  0)
@    239461 SH: first_mb_in_slice                                     1 (  0)
@    239462 SH: slice_type                                        00110 (  5)
@    239467 SH: pic_parameter_set_id                                  1 (  0)
@    239468 SH: frame_num                                          0100 (  4)
@    239472 SH: pic_order_cnt_lsb                                0001000 (  8)
@    239479 SH: num_ref_idx_active_override_flag                      0 (  0)
@    239480 SH: ref_pic_list_reordering_flag_l0                       0 (  0)
@    239481 SH: adaptive_ref_pic_buffering_flag                       0 (  0)
@    239482 SH: slice_qp_delta                                   0001000 (  4)

********** Pic: 4 (I/P) MB: 0 Slice: 2 **********

********** Pic: 4 (I/P) MB: 1 Slice: 2 **********

********** Pic: 4 (I/P) MB: 2 Slice: 2 **********

********** Pic: 4 (I/P) MB: 3 Slice: 2 **********

********** Pic: 4 (I/P) MB: 4 Slice: 2 **********

********** Pic: 4 (I/P) MB: 5 Slice: 2 **********

********** Pic: 4 (I/P) MB: 6 Slice: 2 **********

********** Pic: 4 (I/P) MB: 7 Slice: 2 **********

********** Pic: 4 (I/P) MB: 8 Slice: 2 **********

********** Pic: 4 (I/P) MB: 9 Slice: 2 **********

********** Pic: 4 (I/P) MB: 10 Slice: 2 **********
```

Figure 2.9: Trace file method with simulated GMV = 0.

We can observe that we do not send information at all, but this is because not even the residual for the first MB has to be sent, and then nothing is sent because the mv that will be used to decode will be zero, that means there has been no movement. Now we are going to see the resulting video, we can see in the Figure 2.10 two video captures separated by 20 frames, we can see how the audience remains static all the time.

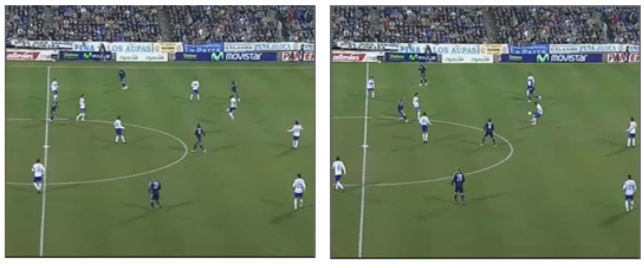


Figure 2.10: Video captures with simulated GMV = 0 separated 20 frames.

After decoding the H.264 file obtained with the decoder, we obtain the same

video file as the encoder (comparing the two videos with a Matlab function are exactly equal), that means the modifications made are correct. This is very important, because the decoder is already standardized, and with this check, we notice the system works properly.

Until now, we have forced the GMV to be fixed and equal to zero, that means that we have checked the easiest option. But as we know, the system has to work with different GMVs, so we are going to test the encoding method with values different to zero, always being an integer. The accuracy of motion compensation is in units of one quarter of the distance between luma samples, that means that a mv equal to four indicates the movement of one pixel.

We are going to check the method forcing the fixed GMV to be four instead of zero as before, we only set to four the horizontal MV, the behaviour of the system in case of the vertical MV is changed is the same. We see the trace_file in the Figure 2.11:

```
*********** Pic: 6 (I/P) MB: 0 Slice: 2 **********

@    212653 mb_skip_run                                              1 (  0)
@    212654 mb_type (P_SLICE) ( 0, 0) =   1                          1 (  0)
@    212655 mvd_l0 (0) =   4  (org_mv   4 pred_mv   0)         0001000 (  4)
@    212662 mvd_l0 (1) =   0  (org_mv   0 pred_mv   0)              1 (  0)
@    212663 CBP ( 0, 0) =   0                                       1 (  0)

*********** Pic: 6 (I/P) MB: 1 Slice: 2 **********

@    212664 mb_skip_run                                              1 (  0)
@    212665 mb_type (P_SLICE) ( 1, 0) =   1                          1 (  0)
@    212666 mvd_l0 (0) =   0  (org_mv   4 pred_mv   4)              1 (  0)
@    212667 mvd_l0 (1) =   0  (org_mv   0 pred_mv   0)              1 (  0)
@    212668 CBP ( 1, 0) =   0                                       1 (  0)

.........
.........

*********** Pic: 6 (I/P) MB: 21 Slice: 2 **********

@    212764 mb_skip_run                                              1 (  0)
@    212765 mb_type (P_SLICE) (21, 0) =   1                          1 (  0)
@    212766 mvd_l0 (0) =   0  (org_mv   4 pred_mv   4)              1 (  0)
@    212767 mvd_l0 (1) =   0  (org_mv   0 pred_mv   0)              1 (  0)
@    212768 CBP (21, 0) =   0                                       1 (  0)

*********** Pic: 6 (I/P) MB: 22 Slice: 2 **********

@    212769 mb_skip_run                                              1 (  0)
@    212770 mb_type (P_SLICE) ( 0, 1) =   1                          1 (  0)
@    212771 mvd_l0 (0) =   0  (org_mv   4 pred_mv   4)              1 (  0)
@    212772 mvd_l0 (1) =   0  (org_mv   0 pred_mv   0)              1 (  0)
@    212773 CBP ( 0, 1) =   0                                       1 (  0)

*********** Pic: 6 (I/P) MB: 23 Slice: 2 **********


*********** Pic: 6 (I/P) MB: 24 Slice: 2 **********


*********** Pic: 6 (I/P) MB: 25 Slice: 2 **********

.........
```

Figure 2.11: Trace file after forcing GMV to be 4.

We can observe how in the first MB the GMV is transmitted (in this case four),

and in the next ones the residual is zero because the real and predicted motion vectors are equal. We can observe that five bits per MB are sent even when Skip mode is set, this is because the macroblocks contained in the first row or column of the image require some bits more because they have no MB reference at their left part, nevertheless the improvement in terms of size is amazing. The rest of the macroblocks do not require the sending of bits at all.

We are going to see now the resulting video. As the GMV is equal to four each frame, we displace the image one pixel per frame. That means after 16 frames we will have move 16 pixels of movement, that means one MB. We are going to see two captures separated 16 frames in the Figure 2.12:

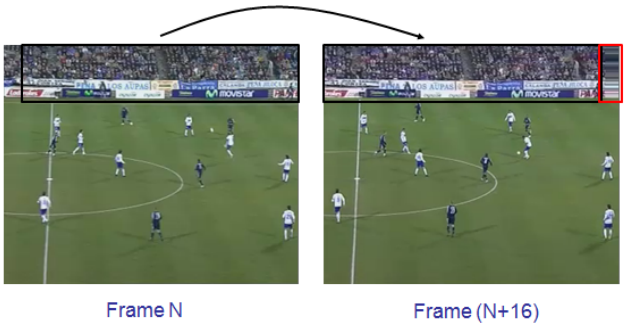


Figure 2.12: Video captures between 16 frames forcing GMV to be 4.

We can observe how we have moved the whole audience directly to the right little by little until having displaced it 16 pixels in 16 frames, this of course is only to check the method, in reality in each frame the GMV used would be different. But there is a problem, how to encode the new information appearing at the border. The macroblocks appearing at the border have no reference in the previous picture to be predicted because that information is completely new. Because of that, we have to preanalyze the video in Matlab to detect the movement of the camera, and when new macroblocks appear at the border (at the left, right or up) we will have to encode these macroblocks with the standard encoding instead of skipping them. To do that a movement detector will me made, that will be seen in the next chapter.

We have checked all is working with zero and four, but we have to check the rest of the values to be completely sure. If we check the encoding method with a fixed GMV of two we can observe some problems. The trace_file is equal to the previous checking with four, but it is transmitted two instead of four. The really important file to observe is the video file, if we look the Figure 2.13, we can observe how the quality of the audience is getting worse very fast after 16 frames, becoming blurry.

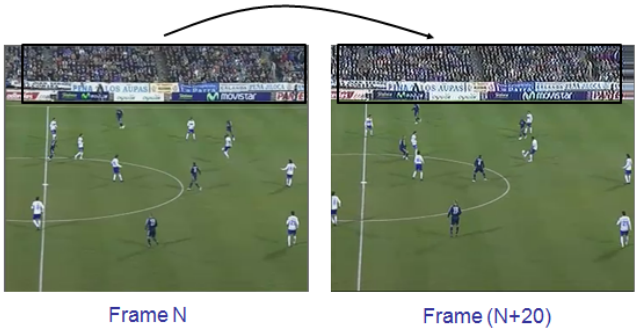


Figure 2.13: Video captures between 16 frames forcing GMV to be 2.

The same thing happens with all the GMVs not multiple of four, this is because of the working of the sampling. If the mv points to an exact sample position, the predicted sample will be that one, but otherwise the corresponding sample is obtained using interpolation to generate the sample. And when we do not send residual along some consecutive frames, as the system is not used to do that, the image starts to get worse. This is for the Luma, but even is worse for the chroma, because the prediction values for the chroma component are always obtained by bilinear interpolation, whatever the motion vectors are.

Because of we are forcing the macroblocks to be Skip along some consecutive frames, the Skip mode is set when no residual has to be sent and the Skip mode has not been created to be forced obviously, together with the characteristic of the sampling, the best thing we can do is using only GMVs multiple of four.

We have achieved to get a new encoding method able to not sending information, but at the same time using one only GMV to reconstruct the image for the

audience exploiting the prediction system of the standard and the Skip prediction mode. Now we have to face the problem of the borders and to obtain the real GMVs preanalyzing the videos with Matlab, in that way we will able to shift the audience to the proper direction according to the values obtained previously.

# Chapter 3

# Preprocessing

From now, we dispose of a new encoding method able to encode a determined slice using one GMV for all the MBs sending a very few bits. Now, we have to make it work, because until now, we have simply used simulated GMVs (equal for all the frames in each case) as a checking of the proper working of the method.

In this chapter we are going to see how to encode videos with the proposed method. To be able to apply a global motion compensation with GMVs, as explained in the previous chapter, we need to have the values of those GMVs. According to the movement in the video, a different GMV will be applied to encode each frame, because of that, a preanalysis of the video has to be done before the application of the new encoding method.

Different video movement detection techniques could be applied, but we have thought about using the information of the movement outcoming of the standar encoding of the videos. In that way, as we can dispose of all the encoding information (including the MVs obviously) by means of the trace file, we can calculate the GMVs as a mean of the MVs of the MBs at the audience in each frame. We are going to preanalyze the videos with the mathematical program Matlab, in that way, it will be easier to measure how much movement it is necessary to be applied, and even in a possible future work, that system of preanalysis could be implemented in the encoder. It could have been directly implemented within the encoder, but the complexity of the project would have been incremented substantially.

Also a motion estimation will be calculated according to the GMVs along consecutive frames in order to address the ourself-called borders problem. That means, how to encode the new information appearing at the borders (because that information is completely new). To solve the borders problem, it will be necessary to include some new modifications in the code of the encoder software, that will be also commented. After applying the method, the real problem of this kind of encoding will be seen, the zoom (explained in details later). First of all, we are going to see the working of the system with we are going to work.

## 3.1 Working of the encoding system

We are going to see a global idea of the working of the system, since we have a new encoding method, we have to calculate the motion information of the camera movement to input that information in the encoder to use it. In the Figure 3.1 we can see the working of the system that we are going to use.

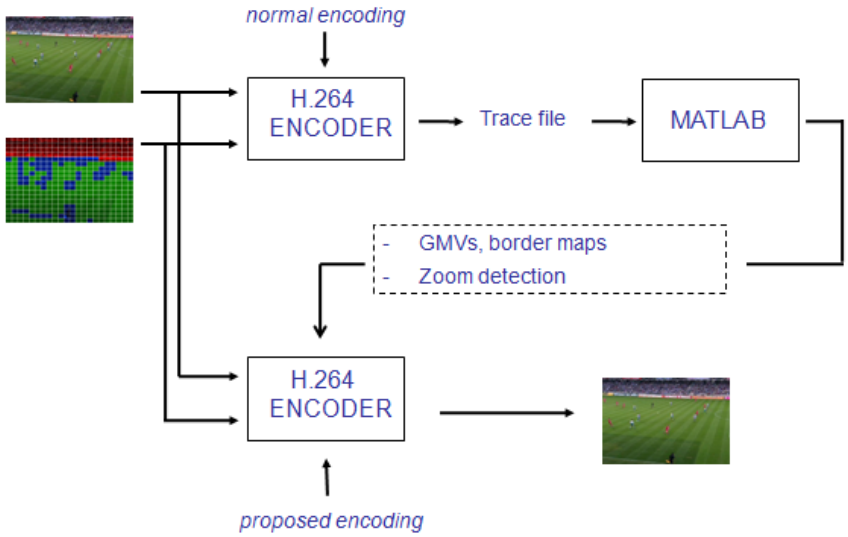


Figure 3.1: Working of the system.

First of all, we are going to encode the video with the normal standard, in that way, we will obtain the trace file containing all the encoding information, being

specially important for us the MVs of the MBs corresponding to the audience according to our allocation Map (already obtained). After that, we are going to input the trace file to Matlab, where the information will be treated. Different kind of information is obtained, about the type of encoding selected, about the global size, the amount of Luma and Chroma transmitted, etc. But at this point we are interested in the motion. We are storing information about the MVs along the video in a variable, which will be used to obtain the global movement of each picture, the shifting applied, the direction, etc.

The first and easiest thing that we have to calculate are the GMVs (one per frame), afterwards we are going to see how to face the borders problem. The main idea is that, depending on the movement detected (camera movement), it is possible to know where is going to appear the new information along the frames. And, modifying shortly the encoder (encode_one_macroblock), we can let some determined MBs when enough movement happens (inputting information to the encoder together with the GMVs) to be encoded with the standard encoding instead of skipping them (applying the GMV).

## 3.2 Preanalyzing the videos with Matlab

After encoding and obtaining the trace file of the video with the encoder in Visual C, we have to make use of Matlab (the popular mathematical program), therefore, we are going to treat all the information of the encoding in an easy way. The trace file is a text file containing wide information about the encoding. Each encoded MB has its coding type, its MVs and its residual, so information about the size, quality and other specifications can be obtained.

We are not going to show the explicit functions and variables of Matlab here, some details about them can be seen in the Appendix, although they will not be explicitly shown. We have modified those functions several times, but only the final versions remain. We will show schemes of the scope of those functions, or only we will comment the tasks made.

### 3.2.1 GMVs Calculation.

We commence the analysis of the video calculating the allocation map corresponding to each frame indicating the correspondence of each MB to the slices. After that, we analyze the trace file and we obtain and store information about the MVs.

We are working with videos in CIF resolution, as we have said in the introducting chapter, that means that we encode each frame with 352x288 (width and height) pixels, so the frames are composed of 22x18 MBs (a MB contains 16x16 pixels). The allocation maps informing of the number of MBs corresponding to the audience and their locations vary in each frame. We store the MVs of the MBs corresponding to each slice along the frames of the video, in that way, we can work with the MVs of the audience. From 0 to a maximum of 16 MVs can be encoded per MB due to the flexibility of the standard. A MV consists of two values, horizontal and vertical (both integer numbers), each value can be positive or negative indicating the direction of the MV. We can see in the Figure 3.2 some possible directions according to the values of the MVs.

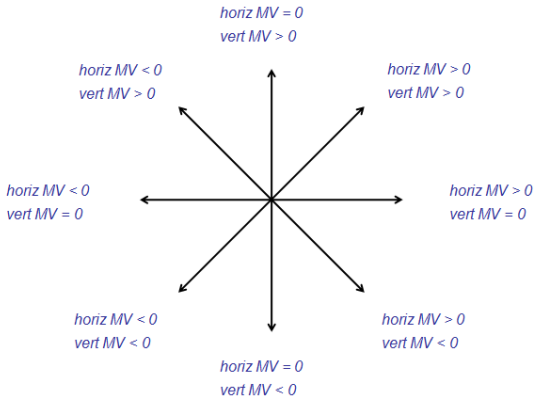


Figure 3.2: Directions according to the MVs.

As commented previously, the resolution of the MVs is a quarter of a pixel, that means a MV equal to 4 means 1 pixel of movement. As seen in chapter 2, according with our investigations and due to the characteristics of the sampling system (interpolation to obtain samples), we only will use GMVs multiple of 4.

The main idea is obtaining the mean of all the MVs contained in the MBs of the audience in each frame to obtain the GMV according to the calculation process (seen in the Figure 3.3), the GMVs cannot be simply the mean of the MBs.

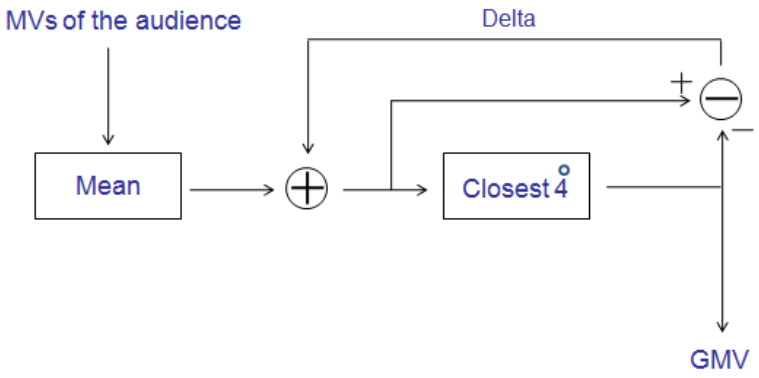


Figure 3.3: GMVs Calculation process.

As we can only use GMVs multiple of 4, we have to round them to be the closest multiple of 4. To avoid to lose fidelity, a delta parameter is used to make more reliable the real values of the GMVs, so the movement that should have been applied is taken into consideration in the next GMVs. In that way the restriction of using only multiple of 4 is not so corruptible. We can observe the worst case that can happen, we can see an example of two consecutive frames in which the camera is moving to the right:

.

Frame N: Mean = 1.9; delta=0; $\Rightarrow$ accumulate = mean + delta= 1.9

temp = mod(accumulate,4) = 1.9

temp $\leq$ 2 $\Rightarrow$ GMV(N) = accumulate - temp = 1.9 -1.9 = 0

delta = accumulate - GMV(N) = 1.9 - 0 = 1.9

.

Frame (N+1): Mean= 5.7; delta=1.9; $\Rightarrow$ accumulate= mean + delta = 7.6

temp = mod(accumulate,4) = 3.6

temp $>$ 2 $\Rightarrow$ GMV(N+1) = accumulate - temp + 4 = 7.6 - 3.6 + 4 = 8

delta = accumulate - GMV(N+1) = 7.6 - 8 = -0.4

...

If we had not taken into consideration the movement that should have been applied in this case, GMVs would have been 0 and 4 (closest multiples of 4 of 1.9 and 5.7), and therefore it would not be very reliable (because the accumulated movement should be 1.9 + 5.7 = 7.6). In that way, one GMV (horizontal and vertical) per frame will be calculated and stored in a text file to be after input to the encoder to be read and applied with the encoding method.

### 3.2.2 Borders problem.

The fact of encoding a frame using inter prediction implies that if some movement happens, the new samples appearing will not have any kind of previous references to be predicted, because of that we think about predicting the movement continually to be able to encode these MBs differently. In our system, we are shifting the content of the audience according to a GMV, but obviously we do not have in the previous picture information about the new one appearing at the border because it is completely new. As there is no reference in the previous picture in order to predict the new samples appearing, the residuals are bigger and contain more high frequency (because the differences between the predicted and the real samples are bigger), that will be bypassed or partially erased. Due to that, depending on the amount of movement happened, normally these MBs at the border need more bits to be encoded and have a worse quality.

According to the direction of the camera movement, the new information appears at the right or left (and up and down also) side of the image. We are going to see the problem that we have to face. We can see an example in the Figure 3.4 of a video moving to the right without taking into consideration the borders problem.

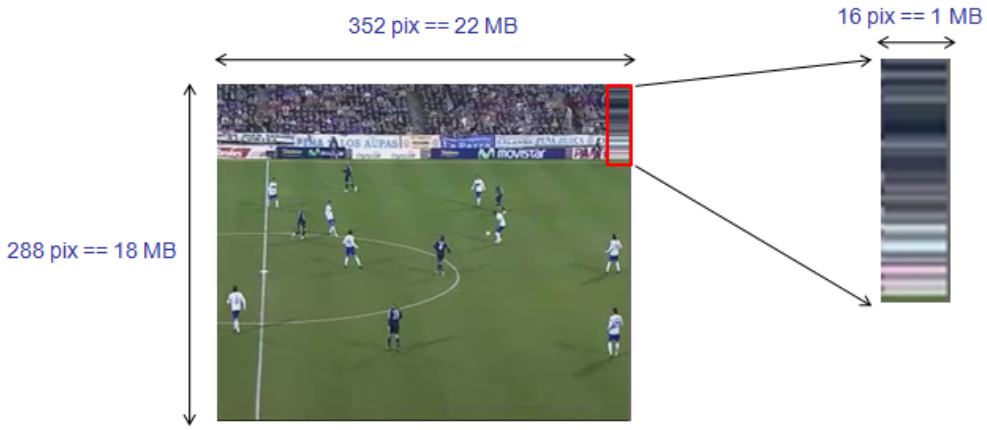


Figure 3.4: Borders problem.

We can observe that, making use of the new characteristic of this standard, the samples in the last MBs at the right border that point over the picture boundaries, are obtained extrapolating the reference picture (replicating the samples of the last pixel at the border). In this case has been shifted the audience 16 pixels (64 MV units), the movement corresponding to a MB. To avoid that annoying visual artifact, it can be thought about encoding all the MBs at he border (according to the movement direction) along the frames with the standard encoding instead of skipping them. To do that we would have to modify the encoder to determine the encoding method. If we made that procedure, and taking into consideration that our main objective is reducing the bit rate, we would not exploit all the possibilities and we would transmit redundant information.

We can think about taking advantage of the preanalysis of the video, as we have to preanalyze the video and input information to the encoder, we dispose of the possibility of inputting the parameters we want to the encoder to influence the encoding decisions. The main objective of our proposal is to reduce the amount of bits sent for the audience. Because of that, we can think about instead of encoding all the MBs at the respective border along all the frames, encoding only those MBs when enough new information should have been encoded in these MBs.

We are going to detect the amount of movement happened in the video along

the frames with the GMVs, we will calculate the addition of consecutive GMVs, and if that addition (in MV units) is higher than a variable threshold, explained later, we will indicate the encoder by means of a text file (also denoted as map) the MBs that have to be encoded normally. To do that, we are going to make a movement detector, a system calculating the motion happened along the frames.

The objective of encoding in that way instead of encoding always the MBs at the border is reducing the bit rate even more, but we have to take into account that the visual artifacts that we will see at the borders do not have to annoy the viewer. So a deal between bit rate reduction and perceived quality has to be done.

We are working, as said before, with CIF resolution, that means 22x18 MBs per frame. As we have to differenciate the encoding of the MBs according with the global movement, we are going to create arrays (one per frame) such large how MBs there are in one frame. These maps will indicate the encoder to use or not our new encoding method in each MB. These MBs maps will be input to the encoder by means of a text file together with the GMVs. Initially these maps will contain values of 0, indicating our encoding method (or 1 indicating the standard encoding).

In the borders maps calculation, a GMVs accumulated addition is calculated continuously in order to control how much new information should have appeared. We compare that addition in each frame with a fixed threshold determined at the beginning. When the accumulated movement reaches that threshold, we encode the MBs at the border normally and we reset the addition, in that way we do not notice too much the annoying border, making the refreshing of the MBs almost unperceptible. The threshold range is between 0 and 64, the range of 16 pixels contained in a MB.

We are going to see an example (with the method implemented already in the encoder, that will be seen later) with a video moving fast to the right using real GMVs with a threshold of 32 (a half of a MB in MV units) in the Figure 3.5.

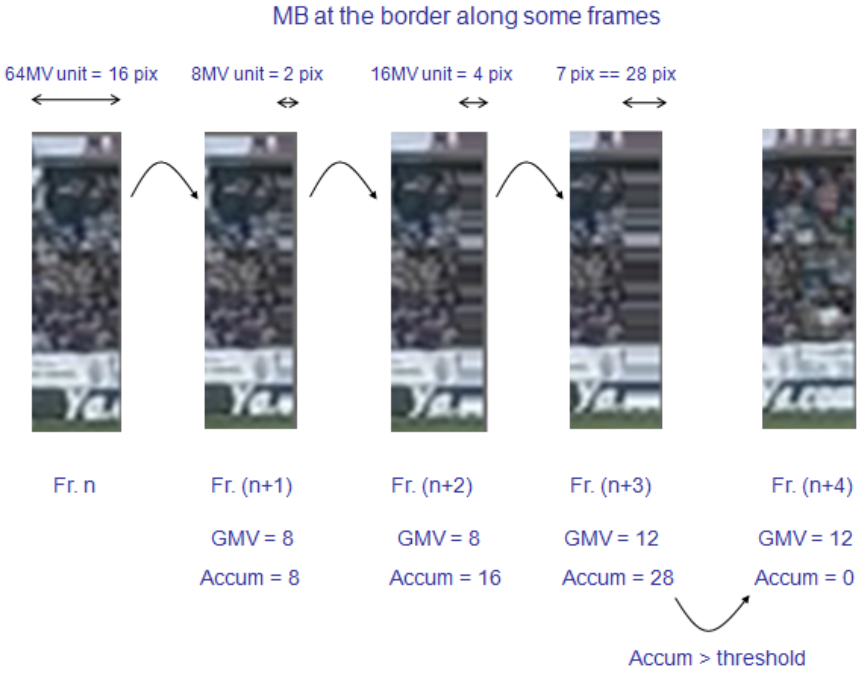


Figure 3.5: Solving Borders problem.

We can see the encoded MB at the right border along some frames and their respective GMVs and accumulated additions. When the annoying information crosses the threshold, we encode those MBs normally (as seen in the frame n+4), and the process starts again. If the movement is fast, that limit is reached in a few frames, and the refreshing is unperciptible, but when slow movement happens along the frames, the annoying border can be noticed, as if during some frames remains the visual effect seen in frame (n+3). Because of that, we detect the situation in which the visual artifact starts to be perciptible and the threshold has not been reached yet. In those situations, when along some frames the accumulated addition remains in a range between the threshold and the threshold divided by two, we encode the border with the standard encoding. If not even the threshold divided by two is reached by the accumulated addition, nothing happens visually (for the visual quality) because that means that only some part of the border of the last MBs are being "bad encoded".

A deal between bit rate and perceived quality has to be reached to define the

threshold. We can see in the Table 3.1 how that threshold affects each factor. After some tests, we have decided that the best option is using a threshold of 32. In that way, when the border starts to be annoying, it is refreshed, without causing perceptible visual artifacts.

| | Threshold ↑ | Threshold ↓ |
|---|---|---|
| **Bit rate** | ↓ (smaller) | ↑ (bigger) |
| **Quality** | ↓ (worse) | ↑ (better) |

Table 3.1: Influence of border threshold.

Summarizing, depending on the values of GMVs (indicating the camera movement), more or less new information appears at the respective border. According to the amount of movement happened and the quantity of frames accumulated, and making use of a threshold, the MBs at the border are encoded with the standar encoding each some frames, in that way, we reduce the amount of bits sent considerably.

Now, we are going to show how we have input these parameters to the encoder, and specially the modifications that we have made in the encoder to act as we have said.

## 3.3 Making the system work

### 3.3.1 Inputting info to the encoder (new modifications).

We have already obtained the tools to make the encoding system works. Now, modifying shortly the encoder we can force the MBs to be encoded with the mode we want, in that way we can (inputting information to the encoder together with the GMVs) encode only the new MBs appearing at the border with the standard encoding.

The first of all, we comment the new variables we have to create, until now, we have created two variables, the variable allowing the encoding method works

and the variable activating the MBs (in case image is P and slice is the audience) to be encoded with the method. We need two variables for the horizontal and vertical GMV, called global_motion_vector[2]. And an array of 22x18 values, called border_map[396], indicating the use of the method or not. We input the GMVs and the maps indicating the encoding of the borders. That information will be read at the beginning of the encoding of each frame, and will be stored in the variables that we have commented with the new created function called read_borders_and_global_motion_vectors_and_zoom_detection (for more information about the modifications see the Appendix).

After having read and assigned those values, we set the real GMVs in the position they should be placed. In these positions we have set before (chapter 2) fixed values in order to test the behaviour of the method. Also we add a new condition to differenciate if the method has to be used or not depending on the border map:

```
if ((audience_skip==1)&&(border_map[img->current_mb_nr]==0))
{
*mv_x= global_motion_vector[0];
*mv_y= global_motion_vector[0];
}
```

We add in all the lines we have changed at MB level (seen in the previous chapter) the condition of the border_map, in that way, if the input border_map indicates the standar encoding (with a 1), then all the conditions will be not true, and the MB will be encoded normally.

### 3.3.2 Encoding the borders.

As we said before, if there is movement, the MBs at the border need more bits to be transmitted and the quality decreases, we can observe an example of this. We see in the Figure 3.6 the result in terms of quality of a standard encoding, comparing the mean quality of the MBs at the border with the mean of all the

MBs. We have made a movement direction detector in Matlab, we can see the discontinuous line with its right vertical axis indicating values above 0 (movement to the right) and below 0 (movement to the left) in units of pixel movement per frame, if the line has a value of 0 means that there is no movement. It can be seen how when movement happens the borders mean quality is below the mean quality.

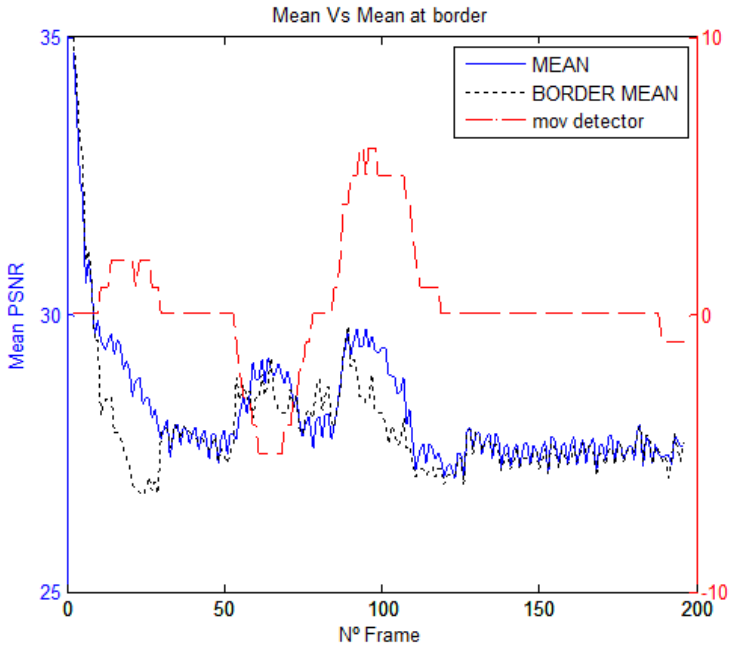


Figure 3.6: Mean border Psnr vs mean Psnr.

The same analysis can be made for the mean size in the Figure 3.7, we can observe how depending on the camera movement the size associated to the MBs at the border is bigger.

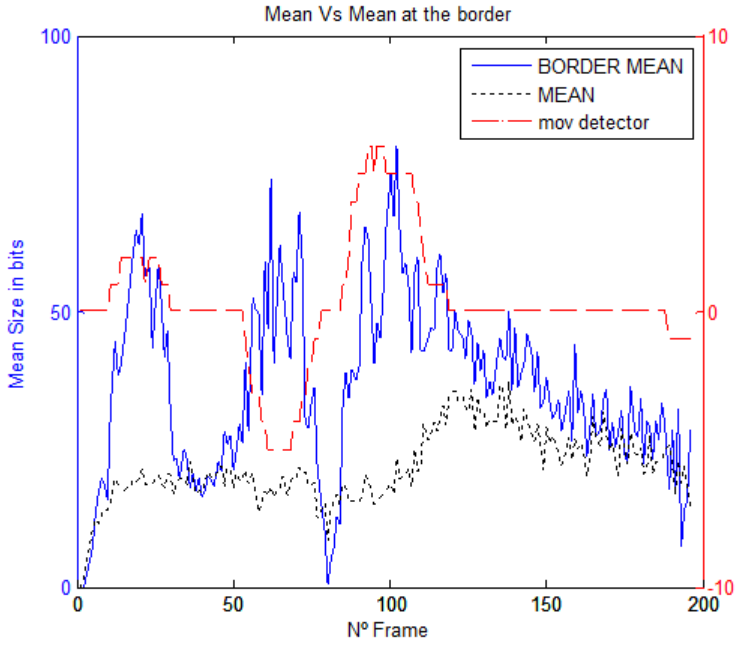


Figure 3.7: Mean border Size vs mean Size.

When inter prediction is used, the MBs at the border (in the direction of the movement of course) require more bits to be sent. Therefore, the idea of encoding those MBs with the standard encoding only sometimes according to the movement, as has been explained before is a good idea to improve substantially the bit rate.

As we are changing the normal working of the encoder, and taking into consideration that we are using inter prediction and we can influence the decisions of the encoding, we can think about the possibility of encoding the MBs at the border as I MBs within the P slice instead of encoding them as P MBs. Usually the I MBs need more bits to be sent (and have more quality), but as we have modified the working of the system, we will have to corroborate it. Also we can think about encoding those MBs at the border with a lower QP, that means a better quality. Within the encoding at MB level, these two possible modifications can be done.

Within the MB level, all the possible encoding modes are analyzed, and the costs associated to each of them are compared between them in order to select the best mode. We can force the cost of the mode we want to be equal to 0 to make the encoder chooses it. In that way, we can think about adding these lines after

the calculation of Intra 4x4 mode, so the MB will be encoded as I MB.

*if ((!intra)&&(border_map[img->current_mb_nr]==1)&&(audience_skip==1))*

*cost = 0;*

Also at MB level, the quality can be increased changing the variable new_qp.

We are going to see the results of encoding the borders as I MBs and with a different quality. We are going to do some tests to measure the quality and size obtained to decide which is the best option to encode the MBs at the border. We will encode the borders with the standard encoding, with the standard encoding with a better QP, forcing them to be I MBs and finally forcing them to be I MBs with a better QP. We obtain these graphs (Figure 3.8 and Figure 3.9) indicating the mean Psnr and the mean size of the MBs at the border (according to the direction) along the frames.

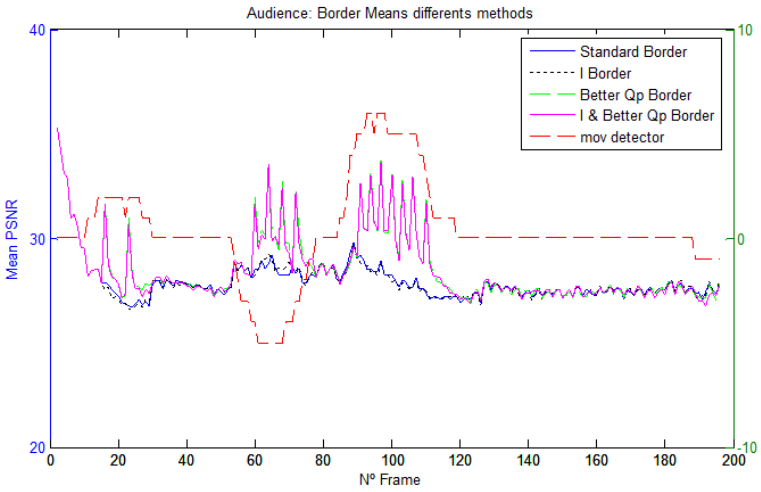


Figure 3.8: Mean border Psnr different options.

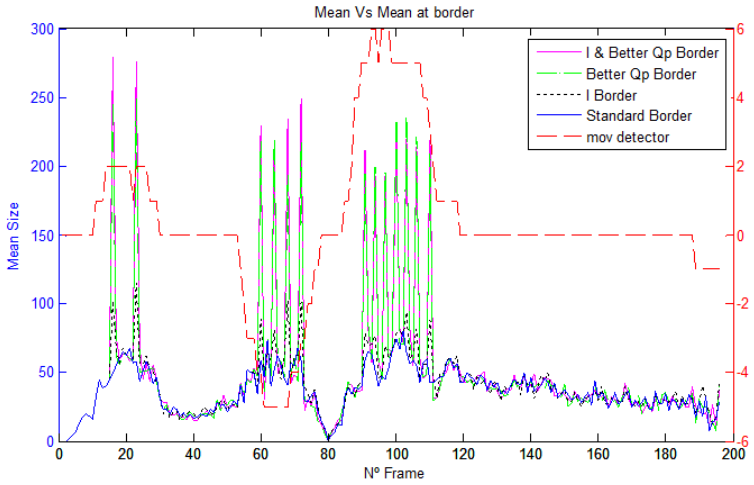


Figure 3.9: Mean border Size different options.

We can observe how the mean Psnr for the standard border and I border options is similar and the same happens for the better QP and I and better QP options. That indicates us that the selection of I encoding does not affect almost the Psnr. The fact of decreasing the QP makes the Psnr increase obviously, but if we observe the size, we can observe how it is considerably increased, because of that we can discard these options. If we observe the size associated to the I border option, we can notice it is bigger than the standard one. So, taking into consideration that we are more interested in reducing the bit rate and the size increasing the QP becomes very big, we decide to encode the MBs at the border without forcing any kind of encoding and without using a better QP, that means choosing the standard option.

We erase the new lines we have added to the encoder to make the tests, excepting the addition of the condition of the border_map in the old modifications to make the system works. In that way, when the encoder reads the map of the borders and it indicates to not to use the method, the encoder will work normally, knowing that the residual sent will be higher than in a normal situation of course.

After having decided how to encode the MBs at the border, we can encode with our method the videos after having input the GMVs and the border maps. At this point we are going to analyze if the system would work or not. To complete the

system we have to see how much time can be used the same reference to predict the next picture. It is to say, we are just shifting the audience according to a GMV, and when new information appears at the border we encode it as new information, but the rest of the picture continues being the same than the beginning but shifted.

As we are using GMV multiple of 4 (explained in chapter 2), the sampling used by the encoder for the luma component does not use interpolation to generate the sample. But, the predicted values for the chroma component are always obtained by bilinear interpolation, whatever are the MVs. Because of that, and because of we are forcing the same MBs locations to be skip along the frames, the colour loses quality step by step, and a refresh has to be done, that means, encoding the audience with the standard encoding each some frames. This refresh implies of course sending more bits, but has to be made. Depending on the refresh time (in terms of frames, not seconds), the encoding will be more or less efficient and the perceived quality. We can see the Table 3.2 indicating how affects the refresh time to each factor.

| | Refresh time ↑ | Refresh time ↓ |
|---|---|---|
| **Bit rate** | ↓ (smaller) | ↑ (bigger) |
| **Quality** | ↓ (worse) | ↑ (better) |

Table 3.2: Influence of refresh time.

After some tests, we have decided to fix a refresh time of 25 frames, that means, without taking into consideration if there has been movement or not, that we will encode normally the audience each 25 frames to update the real audience, that will be shifted later.

At this point we are going to see the big problem we will have to face the next, the appearance of the zoom. As we are going to see in the next chapter, this method does not work if there is zoom, so we will have to detect it, to instead of using our special method, using the standard one.

# Chapter 4

# Appearance of zoom.

The possibilities of video capturing allow us to use different cameras with different features. Soccer videos can contain very different contents. They contain various scene changes, fast and slow motion scenes, zooming (in and out) and wide angle panning sequences. Different camera operations can be used as explained in [16]. We can see in the Figure 4.1 these possible camera operations.

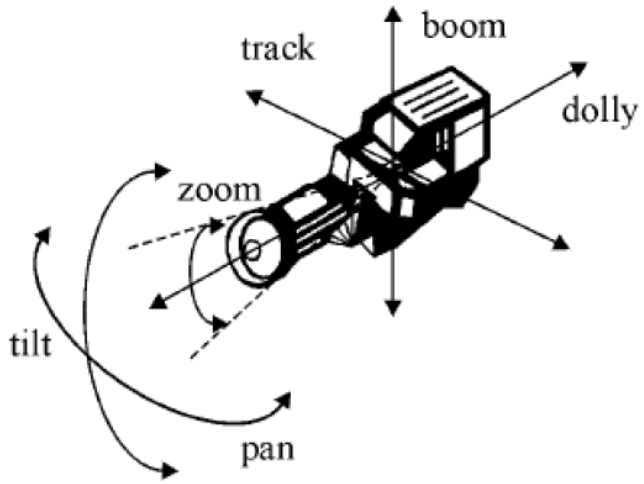


Figure 4.1: Camera operations.

Although different cameras are used, in order to have a general view of the play, normally the sequences in the case of soccer videos are taken with a wide angle camera with the already commented possible operations. Within a soccer

match, each scene is composed of a succession of shots related semantically, that means, a sequence of related pictures captured between scene cuts. Along this thesis we are going to work with wide angle sequences without camera changes, in which zoom is included, which is going to be the object of our investigation.

The zoom is the function or utility that allows capturing pictures from a distant view to a more close-up view (zoom in) and vice versa (zoom out), making the objects appear closer or more distant respectively. Video sequences are captured with digital video cameras including this feature, the optical zoom, using lens with the ability to vary its focal length (and thus angle of view) without losing quality.

The zoom was created to allow us capturing a far object with a very high resolution, bringing a far object nearer. This feature was used for the first time in photography, but it has been extended to be utilized in video capturing for many applications. In the context of sport video capturing, it has been used specially in soccer videos. The play in a soccer video in that way can be seen from a far wide angle point of view or in a more details way making the effect of the camera moving toward some subject.

## 4.1 Main problem and proposed solution.

The appearance of the zoom is one of the limitations of this encoding method. If there is no zoom, the position of the objects in the audience changes according to the movement of the camera (one of the fundamentals of this project) and the whole audience can be considered a static background. But when the zoom is applied, the size of the objects increases or decreases depending on the kind of zoom (it does not matter if there is also movement or not), and therefore the method cannot work properly. If the size associated to each object in a frame changes, it cannot be represented just shifting them as we are doing until now.

Therefore, our encoding method can be applied only if the camera movement does not include zoom. We can see graphically what happens when a zoom appears in the Figure 4.2 after having encoded the video with the standard encoding.


Figure 4.2: Zoom in example.

If we pay attention to the advertisement contained in the audience marked with a discontinuous line, we observe how its size (pixels assigned for its representation) has increased considerably after the zoom along some frames. We can imagine that if we use our encoding method to shift the audience, it would be impossible to represent properly the new audience with the information of the previous picture and the use of one GMV for the whole audience would not be effective.

This encoding method is based on the property of the audience to move toward a direction depending on the camera movement; being the mean size of the MVs an indicator of how fast the overall movement happens. The MVs in the horizontal and vertical directions are tipically parallel and their magnitudes are approximately the same (as seen at the left side of the Figure 4.3), in these cases our system works properly. But if we take a look to the MVs in the case of zooming the MVs do not point in the same direction (as seen at the right side of the Figure 4.3).

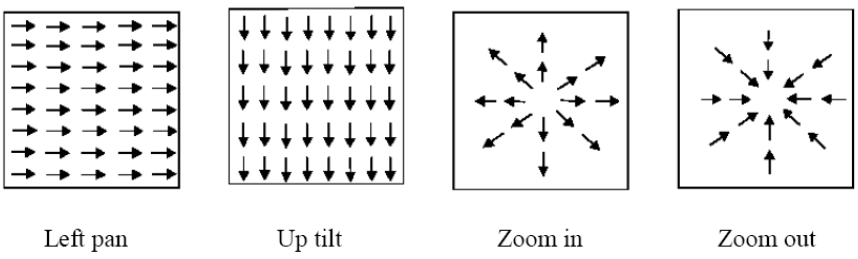


Figure 4.3: MV patterns resulting from various camera operations.

This fact can be checked observing the MVs distribution of the audience. All the MVs are concentred in a region, that means that most of them point in the same direction with similar length, we can see it at the left side of the Figure 4.4. But if we observe the MVs distribution of a frame in which the zoom has been applied, we can observe how the distribution changes. We can see it at the right side of the Figure 4.4 , we can observe how some tails have appeared. That means that all the MVs are not pointing in the same direction, and their values are not concentred in a region.

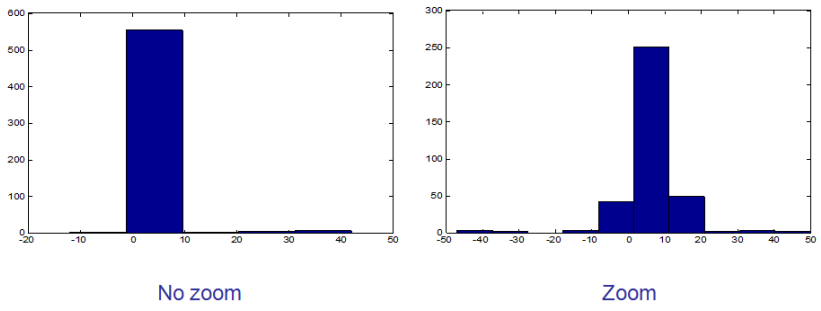


Figure 4.4: Histograms "no zoom" and "zoom".

As the method cannot be applied if zoom appears, we have to propose a solution to avoid to use the encoding method in these cases. We solved this problem realizing a zoom detector. It is able to indicate to the encoder to use the encoding method or not depending on the MVs distribution. As we are preanalyzing the videos with Matlab, we are going to implement a "zoom detector" in Matlab also. As we have said before, we are using information outcoming from the standard encoder. In that way, it could be easier implementing all the preanalysis that we are doing within the encoder in a possible future work if the results were amazing.

Along this chapter, we are going to see the state of the art of zoom detection, to afterwards implement our zoom detector, and investigate the parameters that influence the process.

## 4.2 State of the art in zoom detection.

Several techniques of video zoom detection have been proposed in literature so far. Motion estimation is the main principal of these techniques, widely studied to improve the tracking of objects and effective analysis. We are going to comment different methods based on MVs analysis. As we have seen at the beginning of this chapter, there are several camera operations, that can influence the capture and therefore the distribution of the MVs. Because of that, zoom detection becomes more difficult when it is used at the same time than panning or tilting.

### 4.2.1 Methods based on MVs.

When zoom happens, there is an extension or contraction of the MBs samples. As we can observe in the right side of the Figure 4.3, the MVs of the MBs of the bottom rows and the top rows have opposite signs. The same happens with the columns at the left and right part. These characteristics are exploited to detect the zoom by Zhang et al [17]. Horizontal MVs (in case of left and right columns) and vertical MVs (in case of top and bottom rows) are analyzed in those MBs, and when the both next conditions happen, zoom is declared.

$$|v_k^{top} - v_k^{bottom}| \geq max(|v_k^{top}|, |v_k^{bottom}|) \tag{4.1}$$

$$|u_k^{left} - u_k^{right}| \geq max(|u_k^{left}|, |u_k^{right}|), \tag{4.2}$$

where v means the vertical component and u the horizontal of the MVs.

Another method was investigated and implemented by Dumitras [18],in this case each picture is partitioned into 8x8 blocks to obtain the MVs. Then, according to the orientation of the MVs, the MBs are grouped in regions of angles. The biggest and second biggest regions (according to the percentage of MBs contained) and the standard deviation of the biggest one are used to determine if there is zoom or not, comparing them with fixed thresholds.

### 4.2.2 Method based on the Hough Transform and MVs.

This method is based mainly in the Hough Transform [19]. As shown in [20], after the application of the Hough transform together with MVs analysis, different patterns can be obtained, and each pattern can be associated to a camera operation. After obtaining the MVs, they are mapped to a polar coordinate space by the Hough Transform. A group of lines with point of convergence/divergence ($x_0$, $y_0$) is represented by the curve $\rho = x_0 * \sin(\phi) + y_0 * \cos(\phi)$ in the Hough space. The least squares method is then employed to fit the transformed MVs to the curve. Finally, if the pattern is sinusoidal, there is zoom.

### 4.2.3 Methods based on Decision Trees and MVs.

To differenciate camera operations, a simple, popular and developed technique is the Decision Trees (DTs) method, developed by Patel and Sethi [21]. For shot detection, they suggest a scheme consisting of comparing intensity, row, and column histograms of successive I frames of MPEG video. For the characterization of those segmented shots, they address the problem of classifying shot motion into different categories using a set of features derived from MVs of P and B frames. The central component of the proposed shot motion characterization scheme is a decision tree classifier built through a process of supervised learning. To build a decision tree, a recursive splitting procedure is applied to the set of training examples so that the classification error is reduced. For more information see [21] and [22].

Anyway, all the methods, beside being sensitive to the noise, do not work properly when zoom and pan happen at the same time, making it very difficult to predict.

## 4.3 Zoom detector implementation.

Our approach consists of a zoom detector based on the analysis of the MVs of the audience. The detection of the zoom in a sequence depends on the characteristics of each frame individually. We are going to detect if there is zoom or not observing the distribution of the MVs. We will group the MVs contained in the audience to know if they point in the same direction and have a similar length. In that way we will declare zoom depending on some parameters. We are going to explain the zoom detector working in details below at conceptual level. The final zoom detection function is borders_and_zoom_detection_and_mvs_calculation3.m, where are calculated the GMVs, the border maps and the zoom decision at the same time.

First of all, it would be necessary to encode the video to obtain the trace file in order to obtain information about the MVs, but that part is already available because we are preanalyzing the videos. After inputting the trace file to Matlab, all the MVs values of each frame are stored in data structures. With the help of the allocation map indicating the correspondence of the MBs to each slice we are able to differenciate the MVs contained in the audience from the others, because we know the MVs of the audience have the property of indicating the camera movement (if there is no zoom).

We make use of the zoom detector at frame level, in each frame is determined if there is zoom or not depending on the MVs values of the audience. As we have seen previously we are going to group the MVs into regions with the help of an histogram to observe the distribution. The main idea is that, after obtaining the distribution, we have to decide if the MVs values are very dispersed or not to determine the use or not of our encoding method, and the difficulty of the process will be mainly how to make the decision. To do that, we will make use of some parameters in order to obtain the best result.

To obtain the histogram, we cannot group all the MVs in regions with fixed lengths because it could happen that a "no zoom" could be considered as zoom

(being a false positive) and vice versa (failed detection) depending on the distribution. For example with fixed region lengths, although most of the MVs are grouped in a range of 6 pixels, if one region contained the half of them, and the next one the half also, zoom would be detected, and we would lose effectiveness and reliability.

To detect properly the zoom we act in this way. We take all the MVs contained in the audience, we consider the maximum and minimum values, and we calculate the range, as the difference between both values. With that range, we make the MVs histogram of the whole range, as we can see in the Figure 4.5. In that way, we have available the amount of MVs associated to each value. Now, we have to make a tool charged to make the decision. To do that we make use of a sliding window in MVs units, that sliding window calculates the quantity of MVs contained in the region that is being covered by it. The window is displaced along the whole range, from the very beginning to the end, obtaining the quantity of MVs contained in each position. In each position, with the results obtained, a percentage respect the total MVs is calculated and stored in an array.

We can observe the position in which the sliding window contains the most of the MVs, the position that contains the highest percentage. We take that maximum percentage, and we compare it with a fixed threshold in order to decide if zoom is declared or not. We can see the method visually in the Figure 4.5. In this example we are analyzing the distribution of the horizontal MVs of a frame with a sliding window length of 4 (that means a movement of 1 pixel) and a threshold of 80% (in which zoom is not detected).

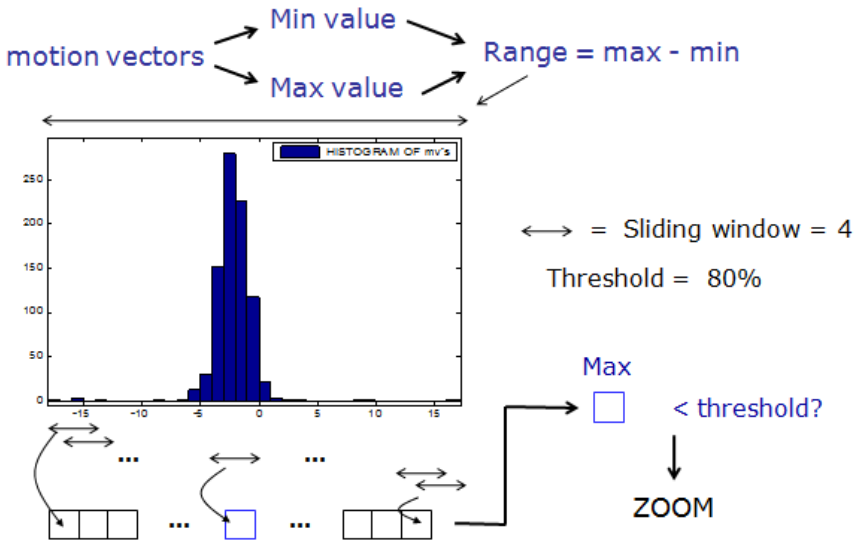


Figure 4.5: Zoom detector scheme.

We are working with MVs, that means that we consider horizontal and vertical components. Because of that we make the analysis that we have shown before for both components, and when some of the two analysis indicates that there is zoom, zoom is declared. Therefore in each frame these two conditions have to be evaluated and satisfied to not to declare zoom.

$$(max(\sum_{i}^{i+w} h\_MVs, \sum_{i+1}^{i+1+w} h\_MVs, ..., \sum_{i+n}^{i+n+w} h\_MVs)/no.of MVs) > thr \quad (4.3)$$

$$(max(\sum_{i}^{i+w} v\_MVs, \sum_{i+1}^{i+1+w} v\_MVs, ..., \sum_{i+n}^{i+n+w} v\_MVs)/no.of MVs) > thr \quad (4.4)$$

, where $n$ denotes the number of windows, $h$ denotes the horizontal component, $v$ the vertical component, $w$ the window length, $thr$ the threshold and $i$ the index of the histogram (starting by the beginning obviously).

We are going to see both examples of distribution (both for horizontal MVs), the one at the left side of the Figure 4.6 is an example of no zoom and the one at the right side is an example of a situation in which zoom has been detected.

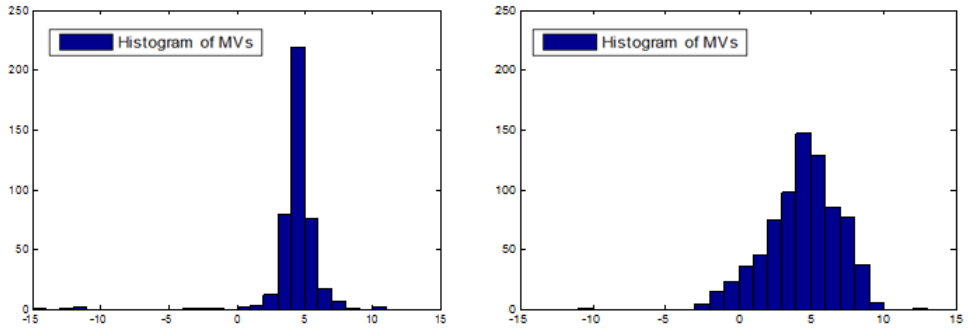


Figure 4.6: No zoom and zoom MVs distribution examples.

We are using a zoom detector based on the cartesian coordinates, although we have also tested a polar zoom detector. In that case, we transform the cartesian coordinates to polar coordinates obtaining modules and angles to work with. We compare the modules and the angles with two different parameters, the first (for the module) similar to the commented sliding window, and for the angles we use a circular sliding window containing a range of degrees. This zoom detector has limitations. When the MVs are concentred near the zero, the values of the angles are very dispersed making impossible its comparison with any threshold. And the module is not so sensitive to variations as the Cartesian coordinates are. Therefore we have decided to use a cartesian zoom detector because the zoom results are more consistant.

We have to fix the values of the two parameters, the sliding window in MVs units and the threshold in percentage. These values influence hardly the decision of the zoom. The zoom searchs the region of dimension "window" in the histogram which contains the maximum percentage of MVs, and then it is compared with the supposed threshold to declare or not the zoom. The threshold indicates the minimum percentage of MVs that should contain the window in a video frame to not declare zoom.

The parameters setting is very difficult because a video capture may provide fixed or variable levels of zoom. The zoom can be progressive or not, slow or fast, it can happen at the same time than panning, etc., and depending on these

characteristics the distribution varies and parameters should be different. And, moreover, not the all kind of videos would work with the same parameters.

Therefore, we can make the detector works more restrictively or less according to the values of the parameters. If there is no zoom and it is declared as zoom the case is called a "false positive" and the opposite case is called "failed detection". We are going to show in the Table 4.1 how the zoom affects normally the two parameters.

| | window ↑ | window ↓ | threshold ↑ | threshold ↓ |
|---|---|---|---|---|
| **False positive prob** | ↓ (better) | ↑ (worse) | ↓ (better) | ↑ (worse) |
| **Failed detection prob** | ↑ (worse) | ↓ (better) | ↑ (worse) | ↓ (better) |

Table 4.1: Influence of window and threshold in the zoom.

So, a compromise has to be reached in order to obtain the best possible detection. At this point we have to think the effect that would have encoding with our encoding method a frame in which zoom has been applied and the detector has considered as "no zoom". In this case, the audience would appear incongruent in the video. And, as we refresh the video each 25 frames (in case no zoom is detected), the refresh would be very annoying and unnatural for the viewer because suddenly the audience contains would change considerably causing a terrible visual impact.

The more zoom the video has, the less bits we will save, obviously. But the visual impact of not detecting the zoom would be even worse. Due to that, we should give priority to detect the zoom restrictively because for the final result is better having more false positives than failed detections. The zoom happens along some frames, not only in one only frame, so normally we have to detect zoom bursts. Therefore, when the detection of the zoom along the frames are inconsistant, that means that the zoom decisions fluctuate between zoom and no zoom consecutively, we will declare zoom.

As there is no a generic tool to know where the zoom is applied, we cannot

evaluate objectively where the zoom detection is properly working. First of all, we analyze video sequences which do not contain zoom, in those situations, the MVs are concentred in a very narrow range. After making some analysis, we have observed that when no zoom is applied most of the MVs are concentred in a range of 4 MVs (1 pixel of movement). To know the threshold that we have to use to make the comparison, we are going to pay attention to the video that we are working with in order to perceive in which frames the zoom is applied. We are working with video captures with different levels of zooming, so that even if a very light zoom is applied, we suppose zoom should be detected.

After previsualizing the videos to know where zoom should be detected, several tests have been done with the zoom detector applying different thresholds and sliding windows. We have decided to set a fixed threshold of 80 % with a sliding window of length 4. That means we are considering zoom if the 80 % of the horizontal or the vertical MVs are not concentred in a range of one pixel of movement. We are going to see an example of a zoom detection with these parameters in the Figure 4.7.

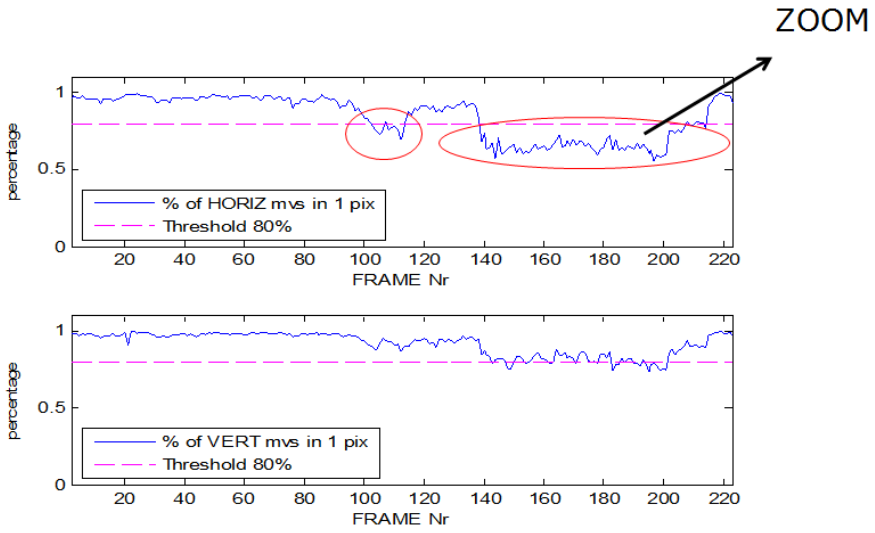


Figure 4.7: Zoom detector example.

We are obtaining the zoom decision for all the frames in the videos, these decisions will be input to the encoder to let our encoding method works in the

respective frames. These decisions will be input to the encoder together with the GMVs and the border maps by means of a text file. They will be read at the same time than the other variables in the same function. A global variable is created to store the decision called "no_zoom", that will be checked at slice level. These are the lines in which the zoom is checked.

```
if ((SliceGroup==2)&&(enable_border_map==1)&&(img->number != 0)&&(no_zoom==1))
audience_skip=1;
else
audience_skip=0;
```

For our objective, the reduction of the bandwidth used for the video transmission, the best can happen is that there is no zoom in the whole video sequence. In that way all the time (except the refresh each 25 frames) our method would be in use, reducing considerably the bit rate. In the next chapter we are going to see the results obtained.

# Chapter 5

# Results

The video test set employed in our experiments consists of seven soccer video sequences with different lengths obtained from a soccer match in which zooming is included. The videos are represented in YUV format in CIF resolution and the frame rate is equal to 25 frames per second (fps) for all of them. The sequences do not contain cuts and they are captured with wide angle camera with different camera operations. Video sequences are preanalyzed as explained previously, obtaining the correspondent GMVs, border maps and zoom detection for all the frames in order to make use of our method. Some already commented factors influence the preanalysis, they have been identified experimentally as explained in the previous chapters. The border threshold has been set to 32, the sliding window length to 4, the zoom threshold to 80 % and the refresh time (in frames) to 25. These values have been maintained constant for all the sequences. In the next Table 5.1 we can see the characteristics of the sequences.

| No | Seq. name | Frame size | No. frames |
|---|---|---|---|
| 1 | m_1 | 352x288 pix | 150 |
| 2 | m_2 | 352x288 pix | 169 |
| 3 | m_3 | 352x288 pix | 127 |
| 4 | m_4 | 352x288 pix | 148 |
| 5 | m_5 | 352x288 pix | 324 |
| 6 | m_6 | 352x288 pix | 263 |
| 7 | m_7 | 352x288 pix | 275 |

Table 5.1: Test set employed.

As we have explained in the previous chapter, we are working with zoom detector parameters that minimize the probability of failed detections because the effect of not detecting the zoom would be very annoying for the viewer using our encoding method. But as we are working with soccer video sequences captured with different camera operations, different levels of zooming are used, and as our zoom detector determines the zoom observing the MVs (comparing consecutive frames), not all the zooms have the same behaviour. When a big zoom happens, it is detected clearly, but the more progressive the zoom is, the more complicated will be the detection. Moreover, when different camera operations like panning and zooming are applied at the same time, the distribution changes, making the detection more difficult.

The traditional manners of analyzing video quality are calculation of mean square error (MSE) and peak signal to noise ratio (Psnr) between the original video and the outcoming encoded video. In this project we are going to use the Psnr to do that. Because we are just shifting the audience without sending residuals, we know that the Psnr is going to decrease considerably. Because of that we can think about measure the perceived quality. Our objective is evaluating the videos in terms of subjective perceptual quality [23]. The Psnr alone is not enough to measure the perceived quality, so we will use a MOS scale, seen in [24].

We have encoded the test sequences with QPs equal to 26 for the players, lines, ball, etc. and 30 for the audience and field. The outcoming videos (together with the trace file) have been analyzed with Matlab to obtain the Bit rate obtained and the objective quality in terms of Psnr. Along this chapter we will see the objective and subjective results as well as its association with the zoom detected.

## 5.1 Objective results.

First of all, we are going to see the objective results, the bit rate saved and the Psnr obtained in comparison to the original sequences.

### 5.1.1 Bit rate.

The main objective of this project is reducing the bit rate applying an alternative method to encode the least important parts (in terms of subjective importance) of video soccer sequences without causing lost of quality perception.

The bit rate saved depends on the amount of zoom detected and on the percentage of MBs correspondent to the audience in each frame. If a sequence does not contain zoom, the method will be applied along the whole sequence (except the refresh each 25 frames) and therefore more bit rate will be saved. On the opposite, if zoom is detected in all the frames, our encoding method would not be used, and no bit rate would be saved. In the other hand, we apply only the encoding method to the MBs contained at the audience, so depending on the number of MBs at the audience, the bit rate saved will be smaller or bigger. Also when the borders are encoded with the standard encoding, more bits are transmitted and the bit rate saved decreases.

We are going to evaluate the effectiveness of the proposed method in terms of bit rate saved. To do that we are going to see the in the Table 5.2 the percentage of zoom detected in each video sequence, the mean percentage of MBs at the audience and the percentage of bit rate saved (comparing the bit rate obtained with our encoding method with the bit rate obtained without using it for the same quality).

| No | % zoom detected | % MBs at the auience | % bit rate saved |
|---|---|---|---|
| 1 | 31 | 17 | 14 |
| 2 | 0.6 | 22 | 50 |
| 3 | 21.6 | 28 | 40 |
| 4 | 14.9 | 23 | 38 |
| 5 | 24.1 | 28 | 40 |
| 6 | 35.9 | 36 | 29 |
| 7 | 46.7 | 26 | 21 |

Table 5.2: Bit rate statistics.

As we can see at the table, the bit rate saved is proportional to the audience MBs percentage and inversely proportional to the zoom detected percentage. We are going to see some graphs obtained as an example. We are going to analyze the results of the sequence number 2, in which zoom has been detected in one only frame. We are going to see the zoom detection graphs (Figure 5.1 and Figure 5.2), the first of them, its calculation, and the second, the definitive zoom decision.

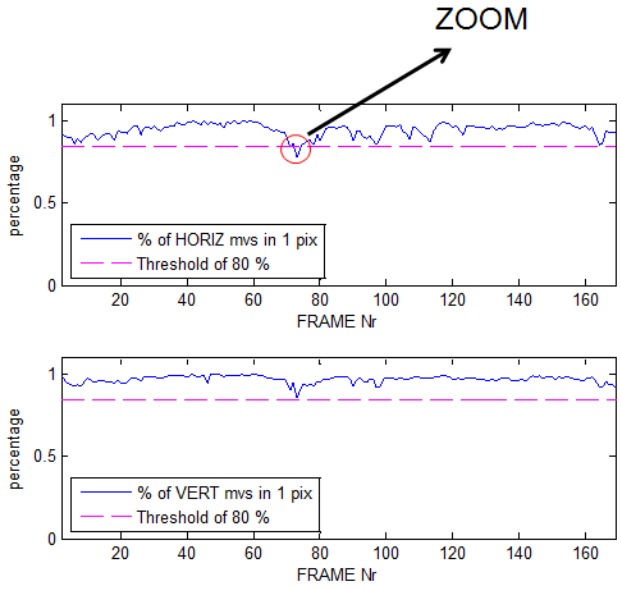


Figure 5.1: Zoom detection calculation for sequence 2.

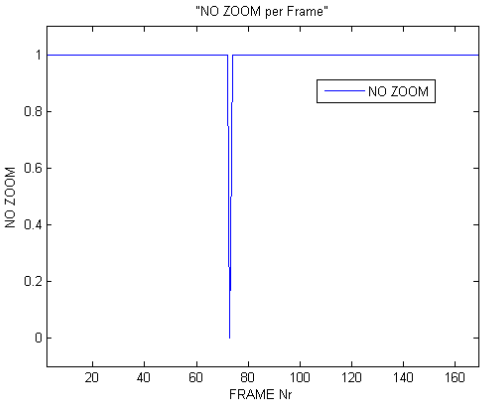


Figure 5.2: Zoom detection result for sequence 2.

Even if no zoom is detected, we refresh the audience each 25 frames, inducing a smaller amount of bits saved. So, although in only one frame has been declared the zoom, we will refresh the audience each 25 frames. Now we are going to analyze the bits transmitted comparing the encoding with and without the application of our method for the same sequence (in which it has been reached a bit rate saved of 50 %). We can see in the upper side of the Figure 5.3 the bits transmitted per frame for the standard encoding (using the method of three slices of course) , and below we can see the bits transmitted with our method. We have represented the bits transmitted for the three regions. The region at the bottom represents the field, the part in the middle represents the players and the part on top represents the audience. We can observe how the amount of bits corresponding to the audience has been considerably reduced. If we pay attention to the peaks appearing, we can think the biggest ones represent the refresh and the smallest ones the borders standard encoding.

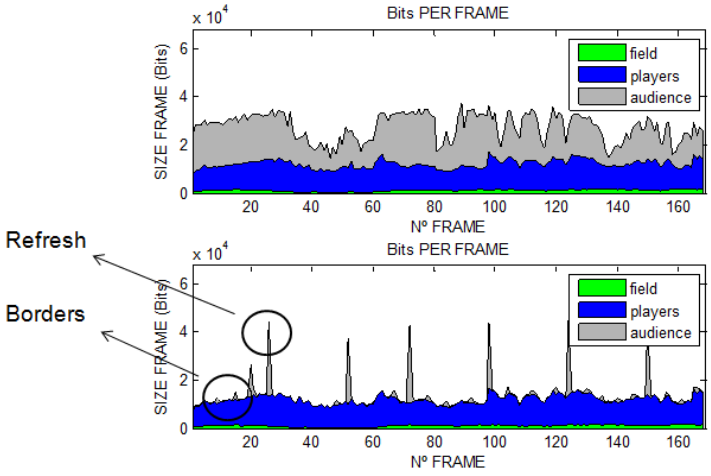


Figure 5.3: Bits per frame sent with and without our encoding method for seq. 2.

In order to have a magnitude of the percentage of the bits transmitted for each zone, we can observe the Figure 5.4. It represents the same information than the previous picture, but normalized.

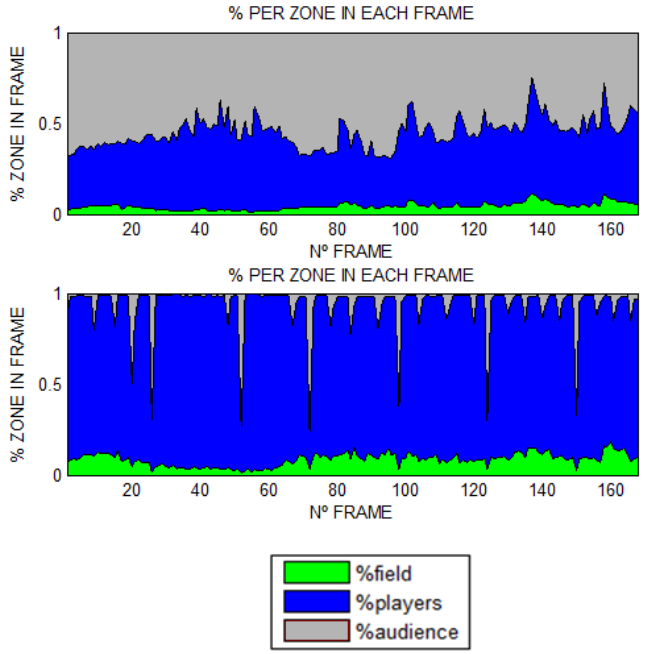


Figure 5.4: Bits percentage per zone with and without our encoding method for seq. 2.

We can also take a look to the mean size of the MBs in each zone. We can observe how with the standard encoding the audience MBs size is even bigger than the players MBs (seen at the upper side of the Figure 5.5), and with our method, we have decreased considerably the assigned size of the audience MBs (seen at the bottom part of the Figure 5.5).

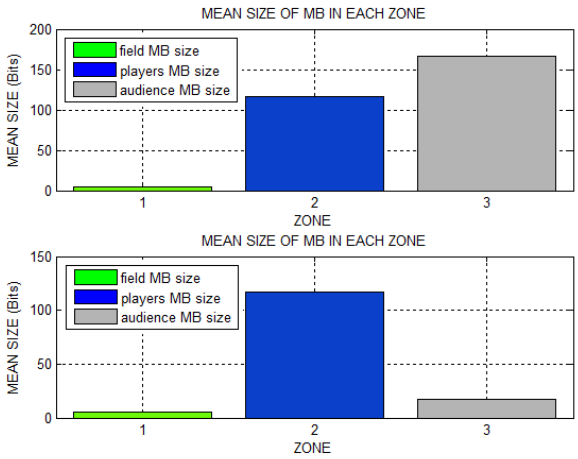


Figure 5.5: Mean MB bits size in each zone.

In the case no zoom is detected, we have achieved to get an improvement in the bit rate around the 50 %, a considerable score. Of course it depends also on the number of MBs of the audience and other factors, but the results in terms of bit rate are amazing. Now we are going to take a look to the quality.

### 5.1.2 Objective quality (Psnr).

We are going to see the objective quality obtained with our encoding method. We are going to use the Psnr as indicator. As we have said before, as we are not sending any kind of residual, and we are using the same picture reference along consecutive frames, it can be thought that Psnr is going to decrease clearly.

The main idea of this project is based on the fact that the audience is the least important part of a video soccer sequence and the part that consumes most part of the bandwidth due to its nature (very high frequency). We analyze the Psnr

for the sequence number 2 and we obtain the next Figure 5.6

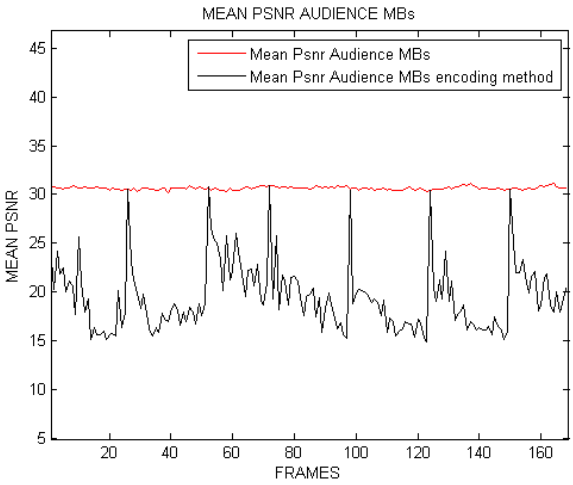


Figure 5.6: Audience MBs mean Psnr with and without our encoding method for seq. 2.

After visualizing the result we can check the fact that we already knew, the Psnr decreases considerably. This has sense because along the frames no updating is being done and we are just shifting the samples. When refresh happens, the Psnr increases, but in a few frames it decreases again. Obviously if there is zoom, the method would not be used and the quality would increase. At this point we can think that the Psnr as a static parameter does not reflect the perceived quality, and because of that we have thought about making a subjective video quality test.

## 5.2 Subjective quality evaluation.

As we want to evaluate the quality perceived, we are going to prepare a quality subjective test using the methodology described in [24]. We want to reproduce a real UMTS video streaming scenario. We are going to simulate an environment in which soccer video sequences will be displayed in a typical UMTS device in CIF resolution.

To prepare the test, we make use of the seven encoded sequences. The sequences which we are working with are representative because they contain different levels

of zooming, zoom detection results and therefore different bit rate saved. With our method we are saving a considerable amount of bit rate. Because of that, if we want to evaluate the subjective quality of our method, we have to compare our outcoming videos with the videos that could be obtained encoding with that reduced bit rate. Therefore, as the standard encoded sequences will require less bits to be encoded, they will be encoded with a higher QP (worse quality) and the comparison will have sense.

First of all, we obtain the reduced bit rates after the application of our encoding method for all the sequences using QPs equal to 30 26 30. Now, with those bit rates, we are going to encode each sequence with a correspondent worse quality according to the bit rate saved. We are working with FMO, (seen at the first chapter), partitioning the frames into three slices, and this is not the standard encoding. Due to that, we are going to obtain and visualize two sequences. The first of them will be encoded using FMO increasing the QP of the slice correspondent to the audience, and the second one (without partitioning using one only QP) will be encoded increasing the general QP. So, for each sequence we have the original sequence, the sequence of the method, the standard FMO sequence and the standard no FMO sequence.

After having obtained all the encoded sequences, we are going to prepare the test. We are going to show in the test the 4 different versions of each sequence. So, the test consists of 28 soccer video sequences. It is not recommended a long test duration because the viewer could not pay attention to the sequences, because the duration of our test is approximately 6 minutes.

All the sequences are displayed one time and in arbitrary order. The sequences are visualized in a UMTS Smartphone Vodafone VPA IV in CIF resolution. After visualizing each sequence, the subjects are asked to rate the video. They have 5 seconds to rate the video until the next sequence appears. The rating scale consists of 5 levels: 5 (excellent), 4 (good), 3 (fair), 2 (poor) and 1 (bad). We can see the pattern of the test in the Figure 5.7.

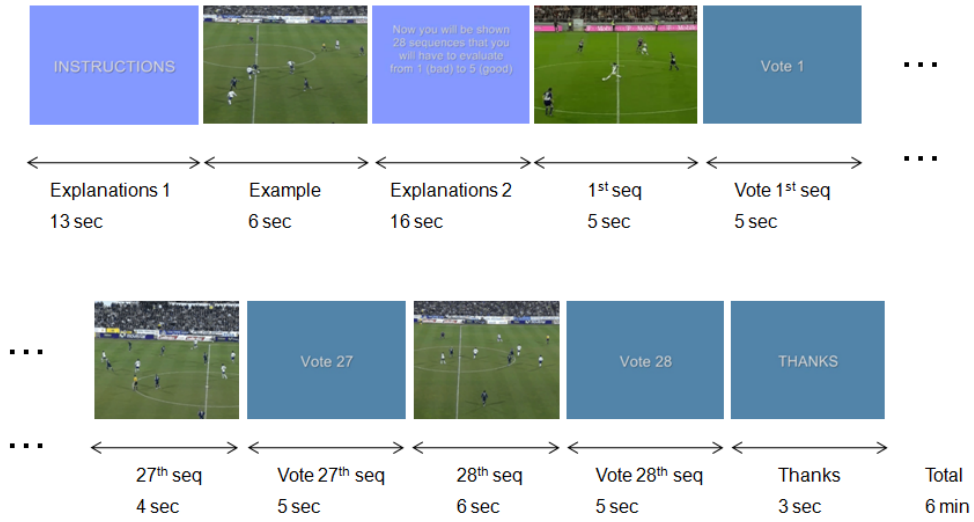


Figure 5.7: MOS test pattern.

The tests have been done by 12 unpaid volunteers. They were women and men with different interests and in a range of age between 18 and 35 years. They have been given a test paper which has been filled up. The videos appear in arbitrary order but we know of course the order in which they have been ordered to evaluate afterwards the results. As it can be supposed, the original sequence should have a high score. And the two standard encoded sequences (FMO and no FMO) should have a different score depending on the bit rate used to encode them. For example, the standard sequences correspondent to the case in which 50 % of bit rate has been saved are supposed to have a low score because they have been encoded with less bits.

We are going to see the results of the video quality survey. We have obtained the mean MOS for each sequence, and we have compared the 4 mean MOS of each sequence separately. We can see in the Figure 5.8.

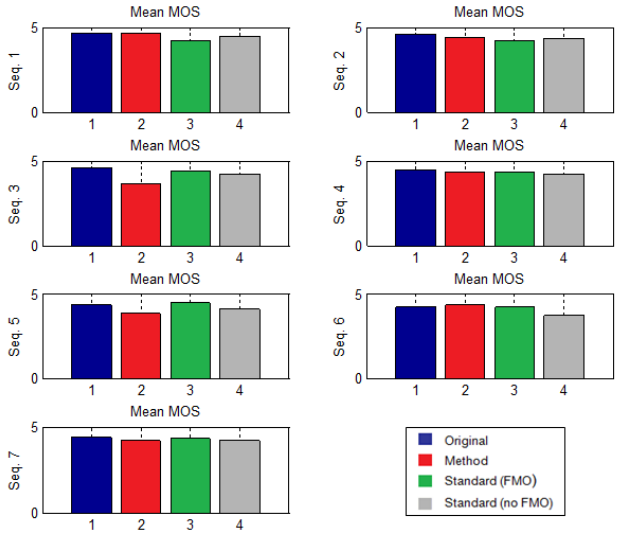


Figure 5.8: MOS test results.

We can see how most of the mean MOS are contained in the range between 4 and 5, that means that the quality perceived is very similar in all the cases. Really, the two standard encoded sequences that were supposed to have a worse perceived quality (because they have been encoded with the reduced bit rate) have been encoded decreasing the quality of the audience. And because really the viewer does not pay attention to the audience details the quality perceived is similar. This fact agrees with the main idea of the project, that is the audience is the least important part, that cannot requires so much bandwidth.

As we have said before, different zooming levels are applied to the sequences. And the more progressive the zoom is, more difficult is the detection. When a zoom is not detected (failed detection) using our encoding method the refresh causes in the viewer annoying artifacts because a big change occurs from one frame to the next one. These artifacts appear in some frames of the test set when gradual zoom is employed, as the case of sequences no. 3 and 5, in which the mean MOS for the second bar appears to be a little bit smaller than the other values.

Some conclusions can be obtained after these results. The test makers do not pay attention to the audience in a soccer sequence, they pay attention to the

play. Only when some artifacts appear at the audience (due to the not efficient detection of zoom) they observe something annoying is happening in the audience encoding. If only sequences without zoom were employed, the zoom would not be a problem and the results would be better. Also the fact of making the tests in a UMTS device with the real resolution makes the viewer to not detect properly if the audience is well represented or not because it has a lot of details, and only the artifacts are noticed.

# Chapter 6

# Conclusions

The objective of this project was to investigate the possibility of modifying the encoding of the audience in soccer video sequences exploiting its characteristic of depending on the camera movement. We have implemented a new encoding method able to encode all the macroblocks (MBs) as skip, but reconstructing them at the both sides (encoder and decoder) with one Global Motion Vector (GMV) transmitted in the first MB of the slice and used for the whole audience. In that way, we have created a method that does not send residuals, but it applies motion compensation.

It can be concluded that it is possible to reduce considerably the bit rate by skipping the MBs of the audience using this method. On the other hand, the objective quality, measured by PSNR, decreases in the MBs of the audience because nothing has been sent. We are only shifting the audience available in the reference picture.

Some effects have to be considered, the encoding of the borders (because the new information appearing has not previous reference), the blurriness (GMVs multiple of 4) and specially the zoom detection. As the method cannot work if there is zoom, we have proposed and implemented a zoom detector in Matlab working with fixed thresholds and parameters. Because of that when gradual zoom happens, its working is not so reliable as it should be. The soccer sequences that have been used in our tests contain different zooming levels, and this makes more difficult

the zoom detection. The more progressive is the zoom, the more difficult becomes its detection.

As the PSNR as a static parameter is not enough to measure the quality perceived, it has been necessary to make a subjective quality test displayed in a typical UMTS device. The carried out surveys results have evidenced the fact that the viewers do not pay much attention to the audience (being the least important part of a soccer sequence) because they pay more attention to the play. Only when the zoom detection has not been successfully performed, some visual annoying artifacts happen in the audience due to sudden noticeable refreshments.

Regarding the idea of extracting a GMV from the field, we have observed that the motion vectors (MVs) of the MBs contained in the field are not so similar as the MVs of the audience are, being more reliable a GMV extracted from the audience. And the application of the method to the MBs of the field could not be realizable because some artifacts would be noticed in the field (where the play happens), being very annoying for the viewer. Moreover the field MB mean size is very low, and its application would not imply major improvements in terms of bit rate.

A limitation of the method is that it can only be used in soccer sequences which do not contain neither logo nor timer in the audience, because they would be shifted along the frames. The next steps to do could be implementing all the things that we have implemented in Matlab directly within the encoder. In that way no preanalysis would be necessary since the whole processing would be performed by the encoder.

To conclude, we can say we have achieved to apply a global motion compensation exploiting the possibilities of the H.264/AVC standard, reducing considerably the bit rate, and obtaining interesting results. This methodology could be applied to other fields of video encoding in the future.

# APPENDIX A

Along this thesis we are working with the H.264/AVC encoder reference software developed by JVT for testing, the Joint Model reference software(JM) [13] version 12.2 (FRExt). The description at functions level has been done previously. In this appendix we are going to show only the lines modified in our project, not the overall code.

## Encoder modifications.

- **global.h**

  *Brief:* global definitions for H.264 encoder.

  ```
  void read_borders_and_global_motion_vectors_and_zoom_detection(void);
  extern int border_map[396];
  extern int enable_border_map;
  extern int audience_skip;
  extern int global_motion_vector[2];
  extern int no_zoom;
  ```

- **void Configure (within configfile.c)**

  *Brief:* Parse the command line parameters and read the config files.

  ```
  else if (0 == strncmp(av[CLcount],"-b",2))
  {  enable_border_map = 1;
  ```

```
CLcount += 1;
}
```

- **void read_borders_and_global_motion_vectors_and_zoom_detection (within configfile.c)**

  *Brief:* Read border maps, GMVs and zoom detection for each frame.

```
void read_borders_and_global_motion_vectors_and_zoom_detection(void)
{ FILE * sgfile=NULL;
int i;
int ret;
int tmp;
int frame_mb_only;
int mb_width, mb_height, mapunit_height;
char NewSliceGroupConfigFileName[FILE_NAME_SIZE];
char file_name[FILE_NAME_SIZE] = "\0";
char borders[FILE_NAME_SIZE] = "\0";
char global_mvs[FILE_NAME_SIZE] = "\0";
if ((input->slice_group_map_type != 0) && (input->slice_group_map_type != 2) &&
(input->slice_group_map_type != 6))
{
    // nothing to do
    return;
}
strncpy(file_name, input->infile, strlen(input->infile) - 4);
sprintf(borders,"./%s/%s_border_%d.map",file_name,file_name,img->number);
sgfile = fopen(borders,"r");
if ( NULL==sgfile )
{
    snprintf(errortext, ET_SIZE, "Error opening slice group file %s",
```

```
    NewSliceGroupConfigFileName);
    error (errortext, 500);
}
frame_mb_only = !(input->PicInterlace || input->MbInterlace);
mb_width= (input->img_width+img->auto_crop_right)>>4;
mb_height= (input->img_height+img->auto_crop_bottom)>>4;
mapunit_height=mb_height/(2-frame_mb_only);
// each line contains slice_group_id for one Macroblock
for (i=0;i<mapunit_height*mb_width;i++)
{
    ret = fscanf(sgfile,"%d", &tmp);
    border_map[i] = (byte) tmp;
    // input->slice_group_id[i]= (byte) tmp;
    if ( 1!=ret )
    {
    fclose(sgfile);
    snprintf(errortext, ET_SIZE, "Error while reading slice group config file (line %d)"
    , i + 1);
    error (errortext, 500);
    }
    fscanf(sgfile,"%*[^\n]");
}
// close file again
fclose(sgfile);
sprintf(global_mvs,"./%s/%s_zoom_and_global_mvs_%d.map",file_name,file_name
,img->number);
sgfile = fopen(global_mvs,"r");
if ( NULL==sgfile )
{
    snprintf(errortext, ET_SIZE, "Error opening slice group file %s"
```

```
    , NewSliceGroupConfigFileName);
    error (errortext, 500);
}
//We store zoom condition
ret= fscanf(sgfile,"%d",&tmp);
no_zoom = tmp;
fscanf(sgfile,"%*[^\n]");
//we store global motion vectors
for (i=0;i<2;i++)
{
    ret = fscanf(sgfile,"%d", &tmp);
    global_motion_vector[i] = tmp;
    if ( 1!=ret )
    {
    fclose(sgfile);
    snprintf(errortext, ET_SIZE, "Error while reading slice group config file (line %d)"
    , i + 1);
    error (errortext, 500);
    }
    fscanf(sgfile,"%*[^\n]");
}
// close file again
fclose(sgfile);
}
```

- **int main (within lencod.c)**

  *Brief:* Main function for H.264/AVC encoder.

```
int border_map[396];
int enable_border_map;
```

```
int audience_skip;
int global_motion_vector[2];
int no_zoom;

if ((img->number != 0) && (enable_border_map==1))
read_borders_and_global_motion_vectors_and_zoom_detection();
```

- **void code_a_picture (within image.c)**

  *Brief:* Code one image/slice.

```
if ((SliceGroup==2)&&(enable_border_map==1)&&(img->number != 0)&&
(no_zoom==1))
    audience_skip=1;
else
    audience_skip=0;
```

- **void encode_one_macroblock_low (within md_low.c)**

  *Brief:* Mode Decision for a macroblock.

```
if ((mode==1)&&(audience_skip==1)&&(border_map[img->current_mb_nr]==0))
{
    cost=0;
}
```

- **int SubPelBlockMotionSearch(within me_fullsearch.c)**

  *Brief:* Sub pixel block motion search.

```
if ((audience_skip==1)&&(border_map[img->current_mb_nr]==0))
{
    *mv_x=global_motion_vector[0];
    *mv_y=global_motion_vector[1];
}
```

- **void LumaResidualCoding (within macroblock.c)**

  *Brief:* Residual Coding of a Luma macroblock (not for intra).

```
if ((audience_skip==1)&&(border_map[img->current_mb_nr]==0))
{
    sum_cnt_nonz=0;
    currMB->cbp=0;
    currMB->cbp_blk=0;
}
```

- **void ChromaResidualCoding (within macroblock.c)**

  *Brief:* Chroma residual coding for an macroblock.

```
if ((audience_skip==1)&&(border_map[img->current_mb_nr]==0))
skipped=1;

if ((audience_skip==1)&&(border_map[img->current_mb_nr]==0))
img->mb_data[img->current_mb_nr].cbp=0;
```

# APPENDIX B

Along the realization of the thesis, some functions have been created in order to be used to preanalyze the videos and to evaluate the results, but only the final versions remain. Now we are going to see the main working of the function charged of preanalyzing the videos, obtaining in that way the GMVs, border maps and zoom detection. We are going to take a look in general to other interesting functions.

## Main Matlab functions.

**borders_and_zoom_detection_and_mvs_calculation3.m**

This function is the main function charged of the preanalysis. This function is called with 6 parameters: file_name, frame_nr, res, window, threshold_border and threshold_zoom, being the parameter cif the resolution, the window the length that will be applied to the sliding window, and the thresholds for the border maps and zoom, already explained along this document.

At the beginning of the function, according with the resolution, the width and height are set. All the structures are prepared to be used. A data base is loaded (being the corresponding data already obtained) containing information about the MVs along the sequence and about the allocation maps. After that, a first frame_nr length loop is passed through, in this first loop, is made the zoom detection as has been explained in the chapter 3. The function eval_zone_mv is called in order to obtain the mean of the audience MVs. After the horizontal and vertical MVs have been analyzed, the decision zoom is declared according to their individual results.

The zoom decision is saved in an array with a length equal to frame_nr obviously called no_zoom. Also if the number of MBs corresponding to the audience varies considerably from one frame to the next zoom is declared because that behaviour is not usual.

After obtaining the zoom decision. A second frame_nr length loop is passed through. The zoom decision is analyzed in this loop, as well as the calculation of the GMVs and the border maps. In this loop is taken into consideration that if along some frames the zoom decision is very changing, the most probable is happening is that zoom is appearing. The GMVs (being only multiple of 4) and border maps are obtained as told in the chapter 3. All the parameters obtained are saved in text files inside a folder with the same name than the video file in order to be directly copied to the bin encoder's folder.

Other functions charged of the evaluation of the results could be also commented in this appendix, but it would not be interesting from because they are simply tools to obtain results.

**Other functions.**

Other interesting functions could be:

- **calculate_psnr_mb** It calculates the psnr of a sequence in comparison with the original, giving the mean psnr per MB in 352x288 format and in 22x18 format.
- **cam_mov** It gives the camera movement according to the GMVs values.
- **quality_vs_size** It outputs results in terms of bit rate saved and quality.
- **border_and_zoom_detection_polares** It calculates the zoom detection with results less amazing than the cartesian detector.

# Bibliography


[1] John Scourias. *"Overview of GSM: The Global System for Mobile Communications"* Published by John Scourias, University of Waterloo, March 13, 1996.

[2] H.Holma, A. Toscala. *"WCDMA for UMTS: radio Access For Third Generation Mobile Communication"* Third Edition, John Wiley & Sons, Ltd., 2004.

[3] ITU-T H.261, Series H: Audiovisual and multimedia systems, *"Video codec for audiovisual services at p x 64 kbit/s"* Mar. 1993.

[4] ISO/IEC-11172-3, *"Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s"* ,1993.

[5] ISO/IEC-13818-1, *"Information Technology-Generic coding of moving pictures and associated audio information: Systems"* ,2000.

[6] ITU-T H.262, Series H: Audiovisual and multimedia systems-Information technology, *"Generic coding of moving pictures and associated audio information: Video"* Feb. 2000.

[7] ISO/IEC-14496-2, *"Coding of Audio-Visual Objects"*, part 2: visual, 2001.

[8] ITU-T H.263, Series H: Audiovisual and multimedia systems, Infraestructure of audiovisual services-coding of moving video, *"Video coding for low bit rate communication"* Jan. 2005.

[9] ITU-T H.264, Series H: Audiovisual and multimedia systems, Infraestructure of audiovisual services-coding of moving video, *"Advanced video coding for generic audiovisual services"* , Mar. 2005.

[10] http://en.wikipedia.org/wiki/YUV

[11] R. Schäfer, T. Wiegand and H. Schwarz *"The emerging H.264/AVC standard"* , EBU Technical Review, Heinrich Hertz Institute, Berlin, Germany, Jan. 2003.

[12] T. Wiegand, Gary J. Sullivan, Senior Member, IEEE, Gisle Bjontegaard, and Ajay Luthra, Senior Member, IEEE *"Overview of the H.264/AVC Video Coding Standard"* , IEEE Transactions on circuits and systems for video technology, vol.13, No.7, Jul 2003

[13] H.264/AVC Software Coordination *"Joint Model Software"* , ver.12.2.

[14] L. Superiori, M. Rupp, *"Encoding optimization of soccer sequences for transmission over UMTS networks"*, International Conference on Multimedia and Expo ICME 2008, Hannover, Jun. 2008.

[15] Yves Dhondt, Peter Lambert. *"Flexible Macroblock Ordering, an error resilience tool in H.264/AVC"*, Fifth FTW Phd Symposium, Faculty of Engineering, Ghent University, 1st December 2004 - paper nr.106

[16] Koprinska, I. Carrato, S., *"Segmentation: A Survey, Signal Processing: Image Communication"*, 16(5), Elsevier Science, 2001, pp. 477 - 500

[17] H.J. Zhang, C.Y. Low, Y.H. Gong, S.W. Smoliar, *"Video parsing using compressed data"*, in: Proc. SPIE Conf. Image and Video Processing II, 1994, pp. 142-149.

[18] Barrz G. Haskell Adriana Dumitras, *"A look-ahead Method for Pan and Zoom Detection in Video Sequences using Block/based Motion Vectors in Polar Coordinates"*, May 2004.

[19] B. Girod, G. Greiner, H. Niemann, *"Principles of 3D Image. Analysis and Synthesis"*, Kluwer Academic Publishers, USA, 2002.

[20] A. Akutsu, Y. Tonomura, H. Hashimoto, Y. Ohba, *"Video indexing using motion vectors"*, in: Proc. SPIE: Visual Comm. and Image Processing 1818, Boston, 1992, pp. 1522-1530..

[21] N.V. Patel, I.K. Sethi, *"Video shot detection and characterization for video databases"*, Pattern Recognition 30 (1997) 583-592.

[22] I.K. Sethi, G.P.R. Sarvarayuda, *"Hierarchical classifier design using mutual information"*, IEEE Trans. on Pattern Analysis and Machine Intelligence, 1982, 441-445.

[23] Olivia Nemethova, Michal Ries and Markus Rupp, *"Quality assessment for H.264 coded low-rate and low-resolution video sequences"*, Published in the proceedings of CIIT,St. Thomas, US Virgin Islands, November 22-24, 2004.

[24] ITU-T Recommendation P.800.1. *"Mean Opinion Score (MOS) terminology"*, July 2006.