INTERACTIVE VIDEO CODING

Aleix Colom

Linköpings Universitet, 2013

Supervisors LiU: Ingemar Ragnemalm, Harald Naustch

Supervisor UPC: Josep Ramon Casas

CONTENTS

Chapter 1

INTRODUCTION

1.1. Project purpose

The aim and development of this project lie in the video coding field, not exactly in the codification theory research but in the practical behaviour of video transcoders. Its purpose is to implement an application to solve one of the common found problems when a video has to be transcoded: finding the optimum coding parameters to get an output video fulfilling all kind of requirements.

One of the possible solutions at this problem is to perform a trial and error method. Given an input video and some requirements (regarding quality or bitrate, for example), a good solution can be found after several transcodifications of the video, changing some parameters each time. This solution is based on watching each transcodification result, identifying the possible existing problems and guessing which parameter can be changed (and how much) to improve the result. Then, a new transcodification is done with these new parameters and the result is studied in the same way as before, repeating the process as many times as it is needed.

Although a good result can be found using this method, there are two main problems that lead to think on other approximations. The first one is the uncertainty of getting a really good solution. Since the parameter changes are based on a human perception about the transcoded video appearance, not all the different possibilities are studied and the actual reasons of good or bad results are hardly known. But even if a good solution can be finally found, this method has a second problem: the time to reach this solution. That is because video transcoding is quite a hard work, in which the codec has to take the optimal decisions after studying all kind of possibilities along the video to ensure that a maximum video quality is achieved using the minimum amount of bytes. Because of this, each transcodification requires a lot of time, which is not a big problem if a single one is needed, but it is in this trial and error approximation, where several transcodifications must be done to finally find a good solution.

Although the method explained above is not the only way to solve the problem, it is useful to understand these two main problems that appear in the process. Therefore, the tool to be developed needs to solve both issues in order to globally perform a good solution to the transcoding process problems. The implemented solution is based on three main points: preview coding, complexity estimation and interactivity. Short sequences are coded several times, instead of transcoding the whole video, paying special attention at the most complex parts and letting the user decide how the final solution will be found.

1.2. Goals

To sum up with the project purpose, three main goals can be defined, as introduced in the last paragraph:

- Preview coding. Instead of transcoding the whole video in order to evaluate the performance of a set of coding parameters, short sequences must be used. Thereby, the first goal of the project is to implement an application which lets the user to select short sequences and sets of coding parameters, and performs a study to find the best ones.
- Complexity estimation. The second goal is to develop an algorithm that provides a complexity estimation of a video sequence. This estimation must let the user see which are the most complex parts of the video and select them to apply there the preview coding.
- Interactivity. Finally, the two previous points must be related keeping a high level of interactivity with the user. The program has to work perfectly fine automatically, but it must give options to the user to select which coding parameters to analyse, which parts of the video to select and what to do at every moment.

1.3. Used tools

To develop this application, two main tools have been utilised to extract information from the input video file, to develop and evaluate the methods and to implement a Graphical User Interface to integrate all the parts.

1.2.1 FFMPEG

The first tool that has been used is FFMPEG, which is actually a whole free multimedia software project. It has analysis and synthesis libraries and programs to handle any kind of multimedia data and it also includes coding libraries and a container format. The project started on 2000 and it is nowadays used in many worldwide audio-visual projects.

In this project, FFMPEG is mainly used to extract visual information from the source videos, to get information from the container format and to make all the needed transcodifications, taking advantage of its command line program. It allows to

externally generate any kind of command and then use it to run FFMPEG functions. It is a quite useful software, supporting any kind of video formats and codecs.

1.2.2. MATLAB

MATLAB is the other mainly used tool. It is a numerical computing and processing application, using its own programming language. In particular, it is commonly used in image and video processing tools and in computer vision, because of its coding and debugging capabilities, although many times the created programs are finally implemented using other languages by the developers.

In the case of this project, MATLAB has an important role in three different aspects. First of all, it has been the programming language used to handle FFMPEG calls and to develop and evaluate all the used methods. Then, it has been rather useful in the image processing part, applying filters and transformations to the input information and processing all the results. And finally, taking advantage of its own Graphical User Interface editor (GUIDE), it has been the used tool to implement the final GUI, where all the developed parts have been integrated.

Chapter 2

BACKGROUND KNOWLEDGE

To understand all the taken decisions during the project development, there is a need to know some fundamentals about video coding. In this chapter, there is a brief review about video coding techniques and formats and also about complexity estimation methods and video quality measures.

2.1. Video Coding

2.1.1. Digital video coding review

All video codecs, although they can have their own aims and particularities, have a common goal: to reduce the weight of the coding files while the video quality is kept as good as possible. If there is not any compression technique applied to the input video, a digital video file will weight too much to be stored or transmitted without problems, hence the trade-off between compression and quality: video quality will be inevitably reduced when the number of bits in the file is also reduced.

Image compression

Since video can be seen as a sequence of a large amount of images, the first step in compression can be the study of how to reduce the amount of bits needed to represent a still image. A raw colour image from an HD video will weigh, at least, 1920 width pixels x 1080 height pixels x 8 bits/pixel x 3 colour images = 5.93 MB, while the same image coded in JPEG format can weigh only about 120 KB, without losing quality apparently. How is this possible?

Firstly, there is no need to keep the value of each pixel in each one of the three colour images RGB. Instead, a transformation can be done in order to separate the image brightness or luma (Y) from the image chrominance (Cb and Cr). This decreases the correlation between the three images and it is proved that chrominance can be downsampled by a factor of 2 or even 4, as it can be seen in Figure 1, without producing significant changes in the image from the point of view of the Human Visual System. This can suppose a first reduction of 50% of the number of bits.

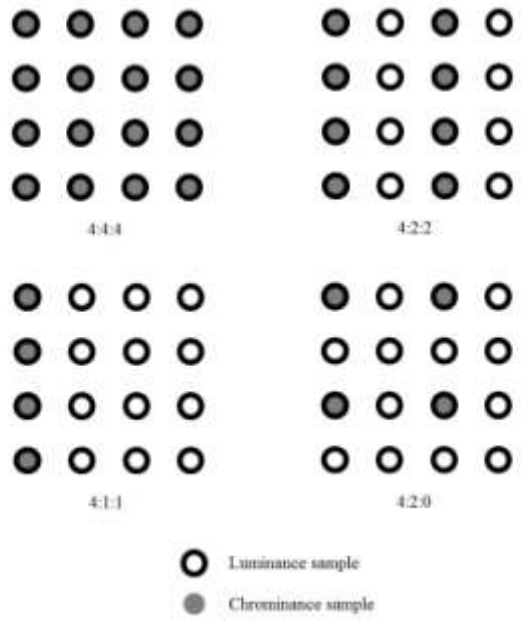


Figure 1 – Different types of chroma subsampling. Chrominance values may be interpolated from the ones before subsampling.

Besides that, there is another source of weigh reduction: taking advantage of the spatial redundancy in the image. This approach is based on the local stationarity in small blocks inside the image. Since usually high frequencies are not present in most of the image, but mainly only where an edge is present, pixel values are very similar if only a few neighbour pixels are taken into account. To take advantage of this spatial redundancy some transformation can be performed to the image, both in the space and in the frequency domains. One of the well-known methods is to split the image in small blocks and apply then a transformation to the frequency domain at each block. It can be done with the Discrete Cosine Transform (DCT), which produces, from 8x8 pixel blocks, 64 new coefficients representing the previous block by 64 different combinations of different frequencies sinusoids. The image can be obtained back without losing any information if all the 64 coefficients are kept.

If the previous assumption was true, only a few of the total number of coefficients at each block will have a significant value: the ones that represent the lowest frequencies. Therefore, if only these few coefficients are kept and the high frequency ones are set as zero (even if they had a small positive value), a high compression ratio will be achieved without losing a lot of information, as seen in Figure 2. It can be done with a quantification matrix.

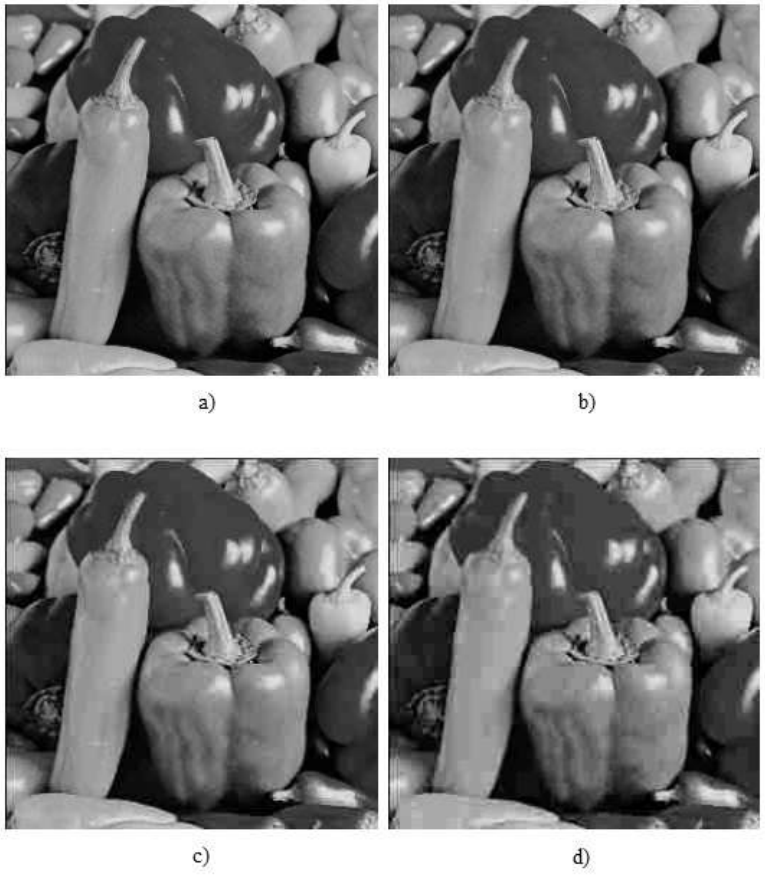


Figure 2 – Quality los by DCT quantization [1]. a) Original image – 100% coefficients. b) 16% coefficients. c) 9% coefficients. d) 6% coefficients.

Finally, how all the coefficients are stored is another way to increase the data compression. The coefficients of the matrix can be scanned in a particular path [2], trying to produce a new array in which the non-zero coefficients are placed in the beginning, whereas the zero ones are placed at the end. Besides that, this array can be coded using an Entropy Coder, like Run-Length, which also reduces the number of needed bits, because of the large amount of zeros in the array.

Image prediction and motion compensation

After reviewing how still images are compressed, the same can be done with video files, which are nothing but several consecutive images. Apart from the space one, there is another important redundancy present in video and it precisely appears when this several consecutives frames are taken into account: the time redundancy. It is obvious that consecutive frames will be similar when it is known that the most common frame rate values are between 25 and 30 frames per second.

Thus, given a single compressed frame, it seems better to compute the difference between this frame and the next one and store this difference (or prediction error), than to store the new frame itself. After that, this second frame can be got back by adding the

stored difference to the previous coded frame. Then, all the following frames can be also predicted from the previous compressed one, producing a higher error when they are further from it. Actually, a frame can also be predicted from a future one, if it is more similar to it than a previous one. It only results in having to decode the future frame first, although it will be displayed later.

What actually happens is that most of the codecs implement all these predictions in a lot of independent consecutive frames along the video, which are called Group of Pictures (GOP), which is visually explained in Figure 3. At each GOP there is a frame which is coded by itself, without taking into account the temporal redundancy. It is called I frame. Then, there are several P frames, which are all predicted only from this I frame. Finally, a lot of frames are placed between I and P frames or between P frames. They are called B frames because they are bidirectional predicted from past and future frames.

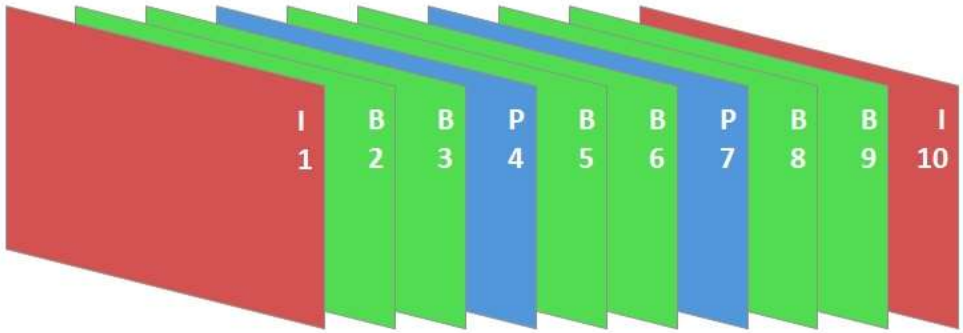


Figure 3- GOP example. Frames 4 and 7 are predicted from frame 1. Frames 2 and 3 are interpolated from frames 1 and 4, and so on. In frame number 10, a new GOP starts.

With this approach, there is a high compression when there are no big differences between close frames. But what usually happens is that, even if the objects in the scene do not quickly change, they don't appear always in the same zone of the picture because they move inside the scene or because of the camera movement. For this reason, most of the encoders use a second tool when a frame has to be predicted from another one: the motion compensation. Splitting the frame to predict in Macro Blocks (MB), each MB is compared to all zones in the reference frame with the same size as the MB, moving it along al the image. When the most similar zone is found, the difference between the MB and the reference found zone is computed, instead of subtracting the MB values from the same position in the reference frame.

It results in having to store the position of the reference zone, which is called Motion Vector, but on the other hand it helps to reduce the difference between a frame to be predicted and the reference one. Because of having the frame split in a lot of MB, each MB can be predicted in a different way from the others. For example, in a B frame, there can be frames predicted either from a previous one, a future one or by themselves, if there is not any similar block in anyone of the references.

Therefore, after this technique is applied, instead of several consecutive frames, there is a single frame to be coded itself and a lot of prediction error frames, which will have lower values than before. In any case, the DCT can be applied to all of them, providing a high compression rate due to all the used tools. Both space and time redundancies are taken into account, in what is called a hybrid encoder, to reduce the amount of bits without losing too much information.

2.1.2. H.264

In the case of this project, a specific coding standard will be used: H.264. It means that it will be the compression format in which the videos will be transcoded and, because of that, all the developed methods to estimate the video complexity will be based on H.264 features. However, due to the similarities between H.264 and most of the previous and further standards also based on a hybrid redundancy reduction, the final implementation can also work if other standards are used.

H.264 is a video compression standard that was completed in 2003. After the previous standards H.262, which is used in DTT, and H.263, which was designed for low bitrate environments, H.264 was thought to increase the video compression efficiency and to provide more flexibility in compression, transmission and storing. This improved compression performance results on higher computational cost to encode the video. However, H.264 does not specify how to encode the video, but only defines the bit stream syntax and the way to decode it. It was implemented by the Joint Video Team (JVT), involving the Moving Picture Experts Group (MPEG) from ISO and the Video Coding Experts Group (VCEG) from ITU. Its name corresponds to the given one by ITU (ISO named it MPEG-4 part 10), but is also commonly known as Advanced Video Coding (AVC).

Intra prediction

Although H.264 does not suppose any great change from past coding standards, it includes some important improvements in the already known coding techniques. The first topic to review is the inclusion of intra prediction in I frames. It means that, in those frames which are not predicted from other ones, it appears an internal prediction of MB in the frame from other already coded MB. Actually, the pixel values are predicted only from the neighbour pixels in the upper and left already coded MB.

Regarding the Y (or brightness) images, it can be done either for whole 16x16 MB or for 4x4 blocks into each MB. In the case of 4x4 blocks, there are 9 different kinds of prediction, depending on the used pixels and the prediction direction. In Figure 4, it can be seen how the information from the reference pixels is used to create the new block prediction.

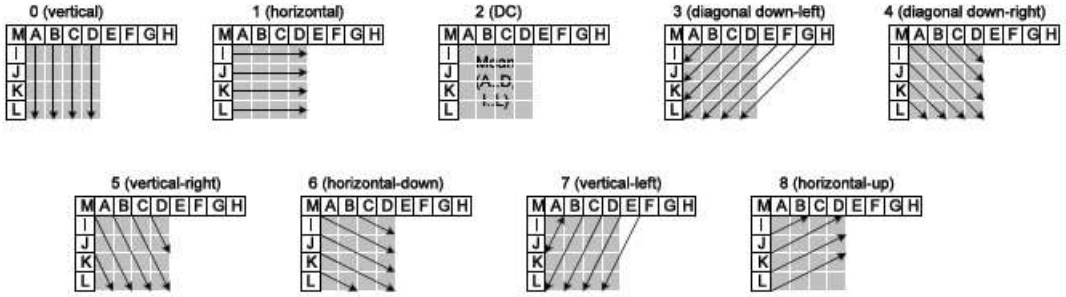


Figure 4 – Prediction from neighbour pixels for the nine intra prediction modes for 4x4 blocks [3].

Besides using 4x4 MB, a whole MB may be also predicted from the neighbour pixels from previous MB. In this case, only 4 different prediction modes are available, instead of the 9 present in 4x4 blocks: from the upper pixels, from the left pixels, as the mean of the upper and left pixels and as an estimated plane from upper and left pixels, as shown in Figure 5.

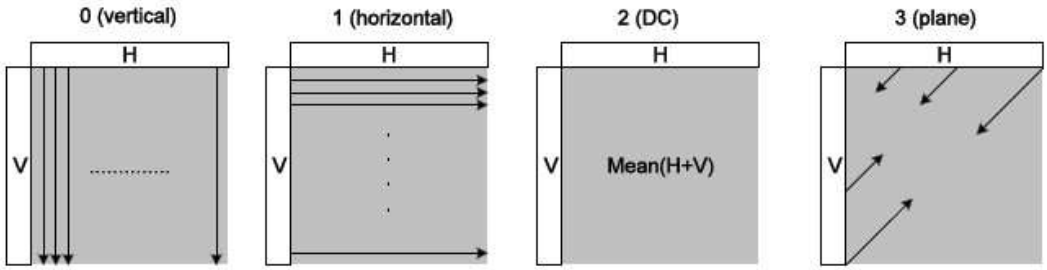


Figure 5- The four prediction modes for 16x16 blocks[5].

Then, an independent prediction is realized at each 8x8 chrominance MB, although both Cb and Cr blocks will be predicted in the same way. Each 8x8 block represents an original 16x16 block, downsampled both vertically and horizontally by a 2 factor. In this case, the same 4 modes as in the 4x4 Y prediction may be selected.

In all the previous cases, the chosen prediction mode is selected after a rate-distortion analysis. Furthermore, taking advantage of the correlation between neighbour blocks, the system predicts the most probable prediction mode, from the modes selected in the previous blocks. It helps to reduce the needed amount of bits in case the most probable mode coincides with the actual best one.

Inter prediction

The second topic to review is the inter prediction: the way in which MB in B and P frames are predicted from I and P frames. In this case, 16x16 MB can be split in more different ways than in intra prediction, to find the one which provides best results. Specifically, 16x16, 8x16, 16x8 and 8x8 partitions can be applied at each MB [4]. Moreover, each 8x8 partition can be also divided into 8x8, 4x8, 8x4 and 4x4 sub-partitions, as it is shown in Figures 6 and 7, providing what it is called tree-structured motion compensation. Regarding chrominance MB, they have the same shape and are

pointed to the same zones as the brightness ones, except that both size and vectors are half of the ones in brightness blocks, due to downsampling.

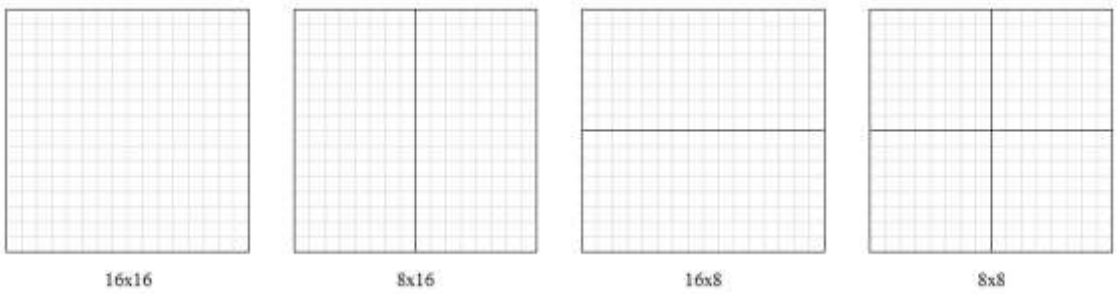


Figure 6 – Possible partitions for 16x16 MB for inter prediction

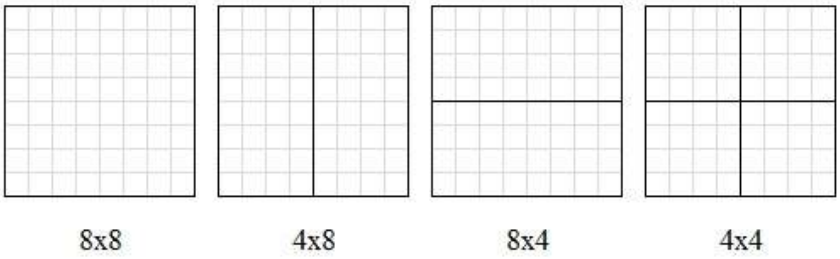


Figure 7 - Possible sub-partitions for 8x8 partitions

The large amount of possible partitions results on a higher signalling cost, but it helps to achieve lower values in the prediction error. In practice, what actually happens is that the most homogenous zones are predicted with larger partitions, while in the most complex zones smaller blocks are needed to reduce the prediction error. In addition, in a similar way as in the intra prediction, Motion Vectors (MV) are predicted from the neighbour blocks, and only the difference with the actual MV has to be stored.

To improve the prediction result, sub-pixel motion vectors are used. This means that the vector values are not integers, with a precision up to 1/4 pixels. Therefore, reference frames have to be interpolated by a factor of 4, because those sub-pixel values do not actually exist.

Other features

Some last features can be described to finalize this review. First of all, H.264 does not actually use a Discrete Cosine Transform (DCT) when I frames or error frames have to be coded. Instead, it uses an Integer Cosine Transform (ICT), which is an approximation of DCT that optimizes the computational cost of all calculations, because only additions and binary shifts are needed when each block is multiplied by the transformation matrix [5].

Besides that, H.264 includes a reconstruction low-pass filter. This filter is applied in the block boundaries after the inverse transformation of the blocks, where

there can appear high artefact gaps between consecutive pixels. This helps to improve the appearance of the decoded frame, as it can be seen in Figure 8.

Figure 8- The same image with (lower) and without (upper) using a reconstruction low-pass filter [6]

Finally, in a higher level, two kinds of entropy coding are used in H.264 to encode all the information (signalling, transform coefficients, motion vectors, etc.): Context Adaptive Binary Arithmetic Coding (CABAC) and Context Adaptive Variable Length Coding (CAVLC) [7][8]. They define some models, with different probability distribution of each bin, to perform the arithmetic codification. Thereby, the coder chooses the model depending on the context (and also adapts the models to the context) to improve the coding efficiency at each particular situation.

2.2. Video complexity estimation

Quality, complexity and bitrate are three highly related concepts in video coding. The more complex is a video, the more bits will be needed to encode it maintaining the same quality. Thus, there is a hard trade-off between bitrate and quality, which is differently solved depending on the context. Usually, bitrate availability defines the coding limits and quality is tried to be maximized within this bitrate boundary, depending on the video complexity. In fact, if the video is coded with Constant Bit Rate (CBR), quality will fluctuate in function of the complexity. Even if a Variable Bit Rate (VBR) is used, there may be too complex to code parts to ensure that a constant quality is reached along the video.

Unlike bitrate, video quality and complexity are two non-well-defined concepts, which are difficult to measure. In this section, some objective and subjective video quality measures will be reviewed, since it is hard to build a model that is able to measure video quality in terms of Human Visual System, which is actually the final target of all audio-visual codification process. Finding a good relationship between subjective and objective measure will speed up the quality evaluation of videos and can be used in the case of this project to estimate the video complexity, since they are highly related.

Objective measures

The first problem when defining objective video quality measures is that it can only be done by comparing original frames with coded frames. This means that the result is highly dependent on the original frames quality and it does not actually represent video quality but only measures the difference or degradation of the coded image from the original one. However, assuming good quality in the original video, the distortion measure of the coded video provides a good representation of the quality loss.

Mean Square Error (MSE) is one of the possible ways to measure the difference of two frames. It averages the sum of the square difference between two images at each pixel. Then, combining the given result of each frame, a single value can be found for a whole sequence. A second approach is to use the Signal to Noise Ratio (SNR), considering the mean of the original frame as Signal and the MSE as error. However, in case of image processing, another method is more commonly used: Peak Signal to Noise Ratio (PSNR). In this case, the highest possible pixel value is considered Signal, while the MSE is considered Noise again. With PSNR, a better approximation to the Human Visual System behaviour is obtained, compared to MSE, but it doesn't actually model how changes in images are finally perceived.

To improve the measure of the difference between two images, Structural Similarity (SSIM) method was implemented. The implementation of all this techniques can be compared in Table 1. This method considers the distortion as the changes in the structural information in the image, whereas previous methods only looked for pixel differences or errors between them. To do so, SSIM value is obtained by calculating mean, variances and covariance matrices of both images.

$$MSE(x,y) = \frac{1}{n}\sum_{i=1}^{n}(x-y)^2$$
$$SNR_{dB}(x,y) = 20\ log_{10}\left(\frac{x}{y}\right)$$
$$PSNR_{dB}(x,y) = 20\ log_{10}\left(\frac{MAX_y}{\sqrt{MSE(x,y)}}\right)$$

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2\mu_y^2 + C_1)(\sigma_x^2\sigma_y^2 + C_2)}$$

Where μ means average, σ means covariance and C are constants to stabilise the division.

Table 1 – Objective quality measures algorithms

Subjective measures

On the other hand, subjective quality measures are very hard to obtain, since they cannot be automatically calculated by any formula. Instead, there is a need to ask people to watch and give a score to each video. However, the obtained results are much more relevant than the ones obtained by automatic methods, because of they actually represent how the video is perceived by the Human Visual System, and there is not any dependence on a previous original sequence.

One of the well-known tests for measuring subjective quality in many audio-visual systems is the Mean Opinion Score (MOS). In the case of video, several sequences are shown to a large group of people. Depending on the aim of the test, that sequences could be either the same scene coded with different parameters or different scenes coded with a specified parameter set, in function of what is wanted to be studied. In any case, all that sequences need to be shown in the same environmental conditions and each one of the evaluators have to give a score between 1 and 5 about the video quality. If there is not any perceptible distortion, a 5 must be given, while if the distortion is completely annoying, the score must be 0. Then a single value can be calculated for each test sequence by averaging the result of each participant.

Relationship between objective/subjective measures and with video complexity

Since it is quite difficult to evaluate video quality by subjective measures, some other objective methods must be found if video quality is wanted to be measured in real time. As introduced before, video quality and scene complexity or criticality are two high related concepts, since given the same bitrate availability, the more complex to code is a scene, the less visual quality is achieved. Thus, it seems that the same approaches can be valid to estimate both quality and complexity.

Although the video coding process involves so many parts which have nothing to do with it, the presence of a lot of objects in a scene and their movement will clearly increase the complexity of a scene. That is because, as a result of the previously reviewed concepts, high frequencies in a frame produce more transformed coefficients to code, and high and fast movements in close frames reduce the capability to predict a frame from previous or future ones.

Because of that, video complexity can be estimated by analysing how much are those two complexity increasing factors present in a scene. For example, some filters can be applied to find high frequencies at each frame and also the changes between

close frames can be estimated [9]. Another approach to estimate the still complexity and the presence of movement can be done from the point of view of the coded stream. For example, the complexity inside a frame can be studied by the size of the intra coded blocks in the frame and the movement can be analysed by looking at how many blocks are forward or intra coded [10]. In the case of this project, the implemented solution is based on the first approach, which is more related to the image processing field, while the second one is closer to the information coding theory.

Chapter 3

PROPOSED SOLUTION AND USED METHODS

3.1. Proposed solution

Since the aim of the project is based on three points (complexity estimation, preview coding and interactivity), the final solution has to contain developed tools and an implemented interface specifically thought to fulfil all the demands of all three bases.

To begin with, a good complexity estimation has to be found. As seen Chapter 2, it seems that there are clearly two complexity increasing sources, which can model both time and space redundancy reduction processes: high frequencies in frames and high differences between frames. What is proposed is to extract some frames along the sequence and process them to get information about the presence of edges and high frequencies and also about the changes between close frames. Then, this must result in a continuous complexity measure along the video, so the most complex parts can be found.

Once these complex parts are found, they can be used in the preview coding part. In this part, short sequences of the video are chosen to be transcoded with different coding parameters, in order to find the ones that accomplish with previous user requirements. That is why finding the most complex parts in the video is so important: if a set of parameters works well in these parts, it would also work well in the rest of the video, while a good behaviour in a random part would not ensure the same in the complex ones. Thereby, in the final interface there must be a part in which user may use the found complex zones to test different sets of coding parameters there.

Besides this, it is important to let the user interact with the program and not to restrict the application execution to an automatically generated solution. The user must be allowed to choose all the possible coding parameters, to ask for transcodifications at any part of the video (and not only at the most complex ones, if he or she thinks that it can be also useful), to do it as many times as desired and to easily see the results. In this case, the user will truly find a way to solve the initial coding problem of large transcoding times and uncertainty of finding a good solution.

3.2. Video information extraction

First of all, some information is needed to be found out of the source video, to be shown in the interface as well as to be used by some of the developed functions: input format, video duration, frame rate, etc.

This information can be extracted from the video using FFMPEG functions by a short processing of the obtained information. The procedure is shown in Figure 9.

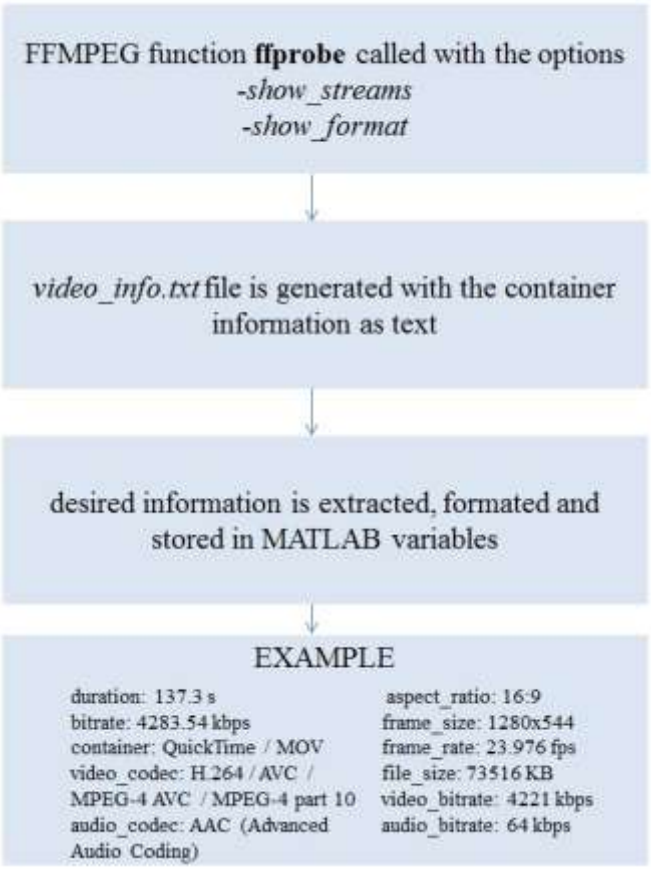


Figure 9 - Video information extraction process description and example

3.3. Frames extraction

To extract frames along the video, a FFMPEG function is used. This function allows to select the exact instant from which a frame must be given along with the frame format and size. Its syntax is described in Figure 10.

```
ffmpeg -ss TIME -i INPUT_PATH/FILENAME -vframes N_FRAMES -s IMG_SIZE OUTPUT_PATH/FILENAME.jpg
          1          2                                    3          4              5
```

Figure 10 – Image extracting syntax. 1) Timestamp of the frame to extract. 2) Input path and video file name. 3) Number of consecutive frames to extract. 4) Extracted image size in pixels. 5) Output path and image filename.

The goal is to get all the desired frames extracted and sorted in a directory to then analyse and process them as it is explained in next sections and Chapter 4. The result is a set of frames that, strategically distributed along the video, give a good representation of the video features and, therefore, can be used to estimate the video complexity. One of the particular goals in the case of the frame extraction is to find the best distribution, which produces good video representation with low number of frames, since it would be the main bottleneck of the whole process, as it will be later explained. Anyway, to show some possible solutions, different frame distributions can be seen in Figure 11.

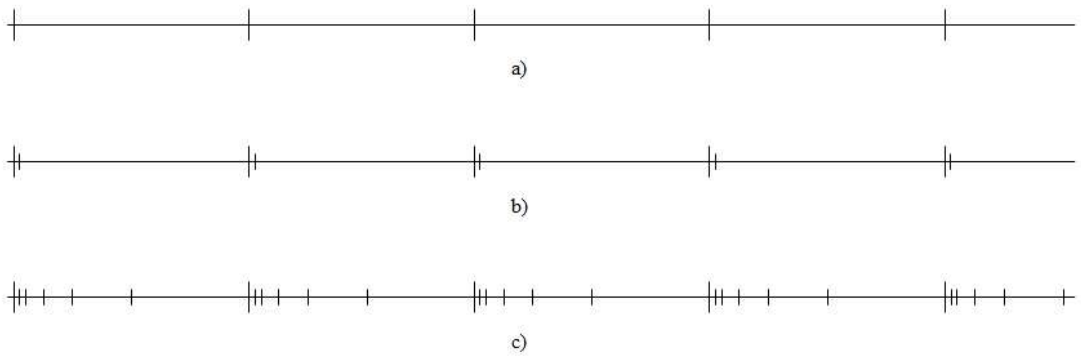


Figure 11 – Three possible distributions of the extracted frames. At each studied point a) a single frame is extracted b) two consecutive frames are extracted c) a set of six frames are extracted to get information from a larger part.

Since a single FFMPEG call is needed for almost each extracted frame, there is an obvious need of a loop implementation in MATLAB. This can be done thanks to the previously found video information by selecting the different instants at each iteration.

3.4. Edges detection

Once all frames have been extracted, it is time to process them, to find both high frequencies and high changes between close frames. In the case of high frequencies, some filters have been tested in order to see which one provides the best approximation of the presence of high frequencies or edges, which will increase the coding complexity. These tested techniques are described below, using the image shown in Figure 12 as an example, while the evaluation process is described in Chapter 4.

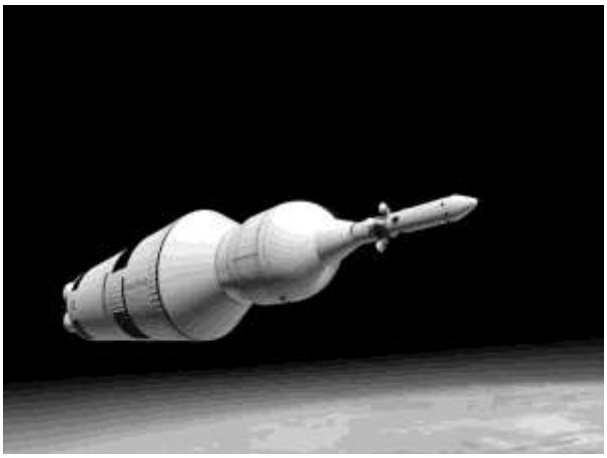


Figure 12 – Example image

3.4.1. Sobel filter

Working as a high-pass filter, Sobel operator is kind of an approximation of the gradient. In this case, a two dimensional 3x3 implementation has been used, as it is shown in Equation 1.

$$h1 = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \quad h2 = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Equation 1 – Two dimensional Sobel filter coefficients.

The result of applying this filter is a new image where high frequency zones are visible: object edges, shadows and complex textures, as it can be seen in Figure 13.

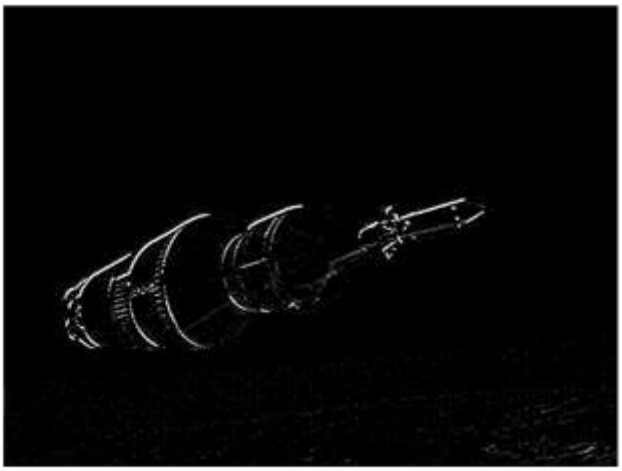


Figure 13 – Result of applying a Sobel filter in an image.

In the case of this project, to quantify the presence of high frequencies, the mean of all the pixels in the result image is calculated.

3.4.2. Prewitt filter

Another well-kown filter in the image processing field has also been tested: Prewitt filter. It is almost the same as the Sobel one, but with a modification of the impulse response, as it can be seen in Equation 2. The result, which is also very similar to Sobel's, is shown in Figure 14.

$$h1 = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \quad h2 = \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

Equation 2 – Two dimensional Prewitt filter coefficients.

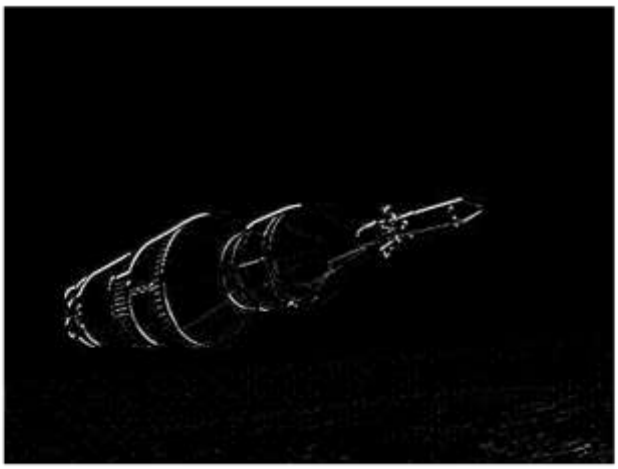


Figure 14 - Result of applying a Prewitt filter in an image

As done before, the mean of all pixels is calculated.

3.4.3. Fourier Transform

Besides classical image processing filters, a method based on the Fourier Transform has also been tested. A Fast Fourier Transform has been firstly applied to the extracted images, and the high frequency regions are studied; in particular, the mean of the values in the four corners, where the highest frequencies are present. The result can be seen in Figure 15.

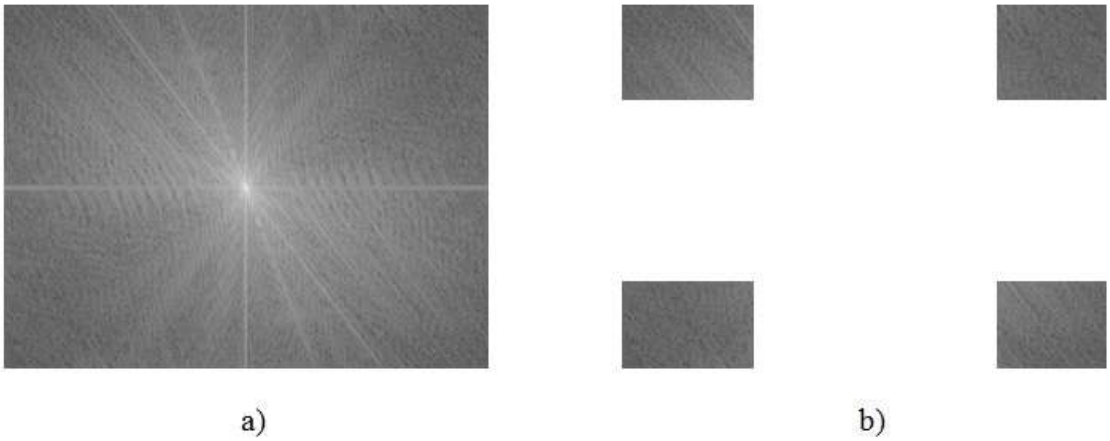


Figure 15 – FFT base method. a) FFT is applied to the image and b) only high frequencies are taken into account. Then, the mean of these values is computed.

3.4.4. MATLAB edge detection functions

Taking advantage of the utilization of MATLAB as the developing tool, some already implemented functions from its toolboxes have been also tried. They are thought to be used in edge detection applications and the result images are similar to the ones provided by Sobel and Prewitt simple implementations seen before. The studied ones are Sobel, Prewitt, Roberts, Laplacian of Gaussian and Canny aproximations. All of them include a decision threshold to see whether an edge will be preserved or not, in function of the values in the neighbour pixels.

In Figure 16, the results of the application of these filters are shown.

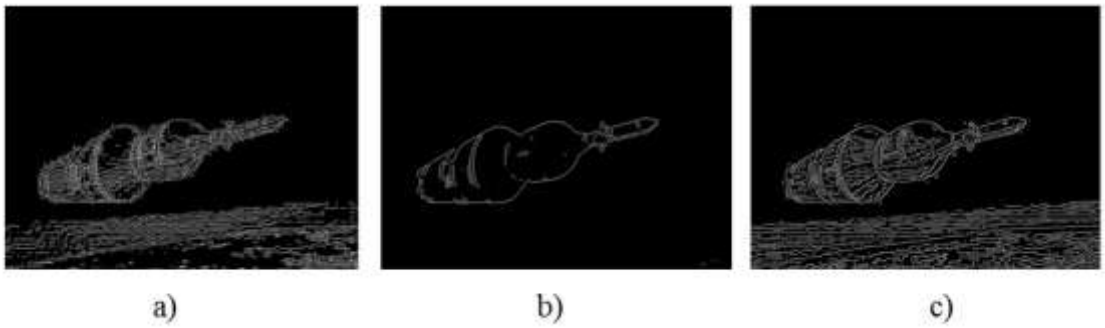


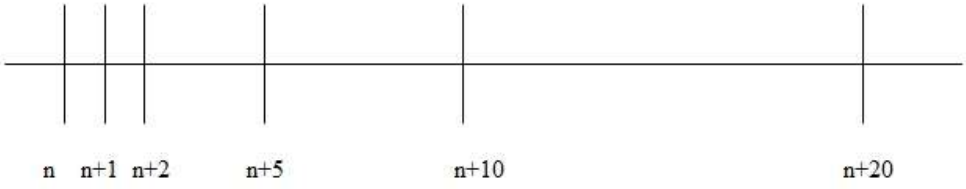
Figure 16 – Result of applying MATLAB edge detection function. a) Sobel version (very similar to Prewitt and Robert ones). b) Laplacian of Gaussian version. c) Canny version.

3.5. Motion detection

Besides looking for high frequencies, changes between close frames must also be estimated. In this case, an image cannot be processed alone because it does not carry

any motion information by itself. Instead, at least two frames must be selected at each time, to try to quantify the amount of movement present from the first one to the second one.

In fact, different combinations of close frames have been tried. At each studied point, some frames have been extracted from a few milliseconds to almost a second from the first reference frame. Then, the difference between all the images, two by two, have been calculated and a single value has been given to each subtraction, by calculating the mean of the resulting image. This process is illustrated in Figure 17, supposing a frame extraction distribution like in Figure 11 c).



a)

M(n,n+1)	M(n,n+2)	M(n,n+5)	M(n,n+10)	M(n,n+20)
	M(n+1,n+2)	M(n+1,n+5)	M(n+1,n+10)	M(n+1,n+20)
		M(n+2,n+5)	M(n+2,n+10)	M(n+2,n+20)
			M(n+5,n+10)	M(n+5,n+20)
				M(n+10,n+20

b)

Figure 17 – a) Supposed frame extraction distribution in a point. b) Resulting matrix, where M(x,y) = mean($frame_x$-$frame_y$)

The evaluation process to select which frame differences are better to be used is described in Chapter 4.

3.6. Global complexity estimation

After all needed frames are extracted and processed, two vectors are available by combining the values at each extracted point: edges estimation (EI) and motion estimation (MI). Webster and Wolf [9] developed an algorithm to give a single value to a whole sequence after processing all frames on it. Although in this project the aim is to obtain a continuous vector to represent the complexity along the sequence, some

methods are based on Webster and Wolf ones. For example, four different vectors are computed before combining them in a single vector: EI, MI, EI·MI and EI', in a similar way as in their algorithm. EI·MI is just the multiplication of both vectors, component by component, to obtain a new vector where the parts in which there is a high value in both EI and MI are emphasised. Then, the derivative of EI (EI') provides another approximation about how much the frames are changing along the video, but in a processed domain in this case.

Then, the best linear combination of these four vectors has to be calculated, in order to find the best approximation of the complexity along the video, in a single vector. This will be explained in Chapter 4.

3.7. Graphical User Interface

The Graphical User Interface will suppose the final integration of all the developed parts in the project. Therefore it is responsible of providing solutions to fulfil the three main goals of the project: preview coding, complexity estimation and interactivity.

Although the whole development process will be explained in Chapter 5, a first scheme of the proposed appearance can be seen in Figure 18. First of all, the upper part will be used for the main menu and to show information, and the main coding options will be introduced in the left section. Then, preview coding and complexity estimation implemented parts are represented in two clearly separated sections. In the complexity estimation section, user will be able to ask for the estimation and to visualise the results by looking at the shown graphics and even at each extracted frame. Then, in the preview coding part some parameters can be chosen and the most complex found parts can be used to apply there the transcodifications. This is where the interactivity appears, because user will be able to decide how all the process will be done, by deciding how many transcodifications will be performed, in which points and with which coding parameters. The resulting previews will be shown in this part and user will be able to play them at any time.

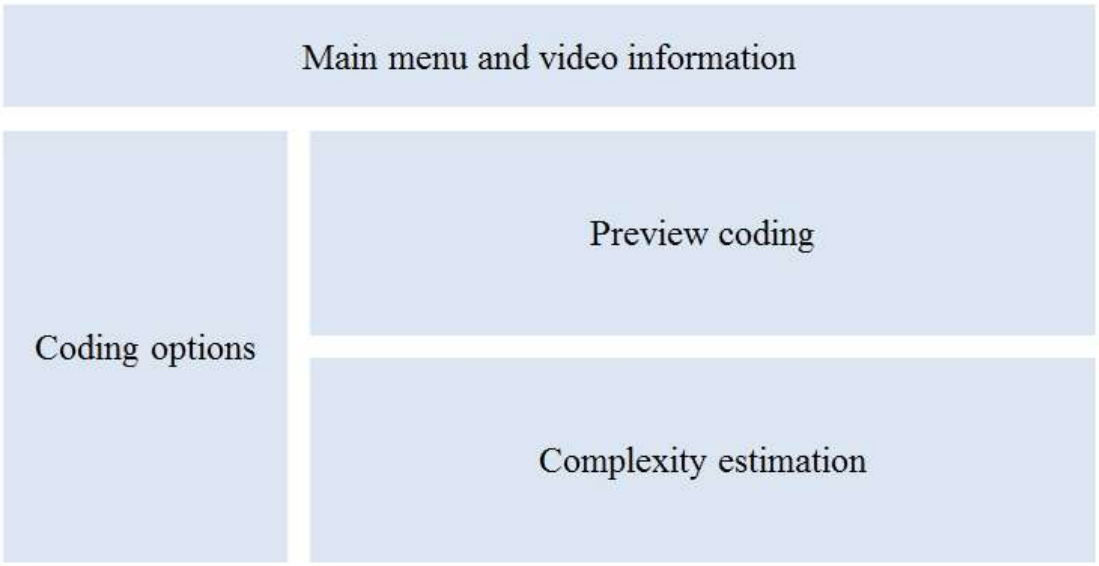


Figure 18 – Schema of the final GUI.

Chapter 4

METHODS EVALUATION AND FINAL IMPLEMENTATION

4.1. Introduction

In this Chapter, the methods presented in Chapter 3 are optimized and evaluated. First of all, the reference obtaining process will be explained. Then the optimization of edge, motion and global complexity estimation will be described. Finally, the global system behaviour will be analysed, both for the full system and for a fast implemented version, which will also be explained.

The system development was mainly thought to work with short videos, from one minute to fifteen. However, a further implementation for longer sequences will be totally based on the described methods and procedures. Since short videos are used, an arbitrary value of 100 studied points along the videos has been selected. It has been studied that it provides a good temporal representation for videos of this kind of duration. This single 100 components vector will be the final complexity estimation, and it will be obtained by combining both edge and motion information vectors.

What will be explained in following sections is how the information is extracted at each one of these 100 studied points to obtain two vectors with edge and motion information. Several frames will be extracted at each point and they will be processed, generating several values. These values are the ones that, properly combined, will provide a single value for each one of the two wanted vectors.

4.2. References

When the complexity estimation process has to be evaluated some references are needed to compare with, in order to know how close the estimations are to the real video complexity. However, video complexity is not a well-defined concept and it is hard to know how those references can be obtained.

Therefore, an experimental method has been chosen to find these references. First of all, ten video files have been selected. There are both real and synthesized videos, from movie trailers, short movies and engineering rendered videos. Then, a single complexity estimation vector has been found for each video, following the below described steps:

1. The video is transcoded using FFMPEG with a higher variable bitrate.
2. Frame bitrate is extracted using FFMPEG functions. The resulting vector is plotted with the value of all frames along the video. The goal is to obtain a bit distribution which shows a higher bitrate in the more complex zones. If the distribution is approximately flat or clearly not connected with the a priori most complex zones, a new bitrate is chosen and step 1 is repeated.
3. Frame types (I, P or B) are extracted using FFMPEG functions. GOP sizes are calculated and the previous vector values are averaged at each GOP. This is done because bitrate vector has high changes between close frames, since I frames bit cost can be over twenty times higher than B or P frames one in the same GOP.
4. A low pass filter is applied to the new vector. This reduces the flatness of large GOPs zones and produces a new vector which is easier to be compared with the obtained complexity estimation. Filter impulse response size has to accurately be chosen in order to remove low undesired fluctuations but keep separated the different complex zones.
5. The final result is exhaustively compared with the displayed video, in order to identify the higher values in the vector with complex scenes in the video. Obviously, Human Visual System cannot know how complex to code a scene is for an encoder, but it helps to identify bad bitrate distributions due to other problems. If the result is not good enough, all process is repeated from step 1.

These steps are repeated for each one of the reference videos, obtaining ten complexity references. The process is illustrated in Figure 19. It has to be said that these references do not exactly represent video complexity, but are adapted to be compared with the complexity estimation.

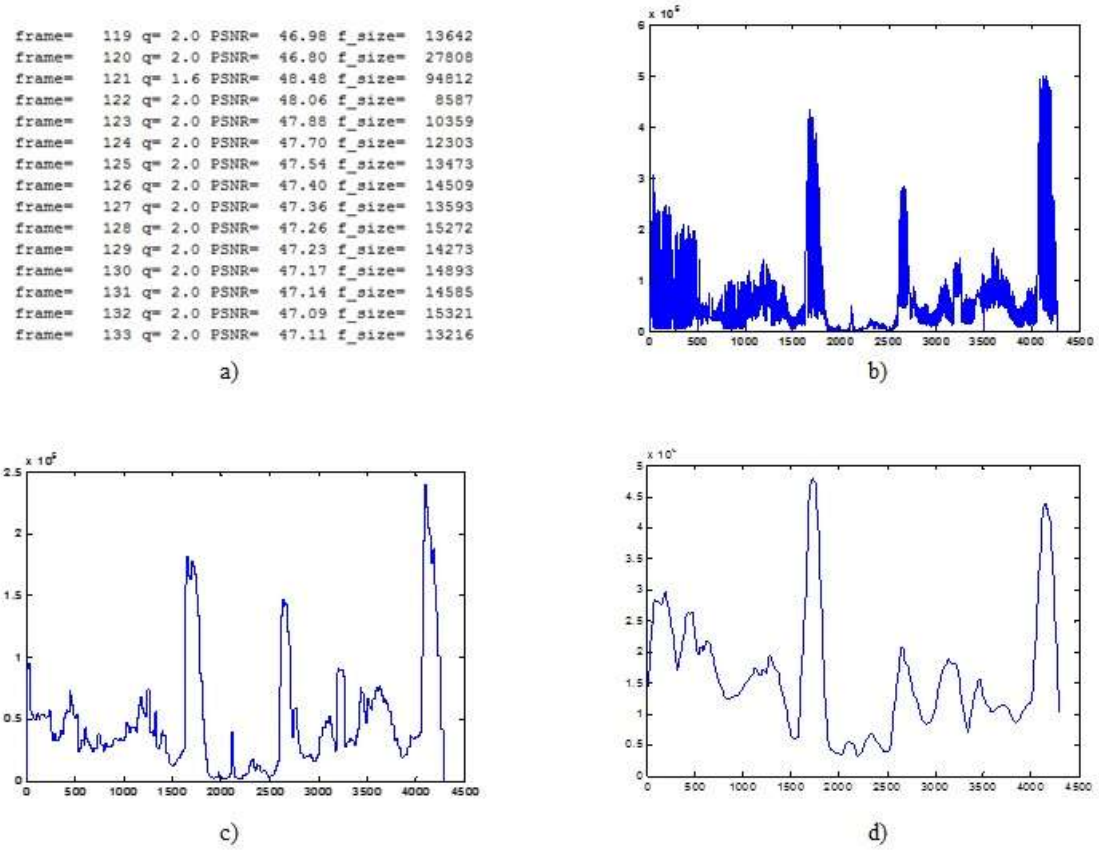


Figure 19 – Reference obtaining process. a) Frame bitrate extraction. b) Bitrate plot. c) GOP averaging. d) Low pass filtering.

4.3. Edges estimation

4.2.1. Evaluation method

In this case, simple references can be used in the evaluation process. Since JPEG format also uses a DCT approach to encode still images, the file size of JPEG images directly gives an approximation of the image complexity. Therefore, applying the before described methods and comparing the obtained result to the size of the images can be a fast evaluation process. To do it, 2000 still images have been tested with all methods described in Chapter 3 (Sobel, Prewitt, FT, MATLAB Sobel, MATLAB Prewitt, MATLAB Roberts, MATLAB LOG, MATLAB Canny) to compare the results with their actual file size.

However, the dynamic range of the file sizes values in bits was highly different from the results given by all methods. This makes the comparison very hard, since changes in the values will not mean the same in the two vectors to compare. Normalizing them by the mean and the standard deviation could be a solution for that kind of problems, but in the case of this project, the given results were not good enough yet.

For this reason, another comparison method was implemented. Considering each vector of N samples, a new value from 1 to N was given to each component, in function of the order position if all the components of the vector were sorted by its value. An example can be seen in Figure 20. If this procedure is applied to both vectors, the new dynamic range is the same for them (since they both had N values) and the comparison can be performed. Obviously, it changes the actual values of the vectors, but it keeps the relative value within each vector and that is what is actually wanted to estimate.

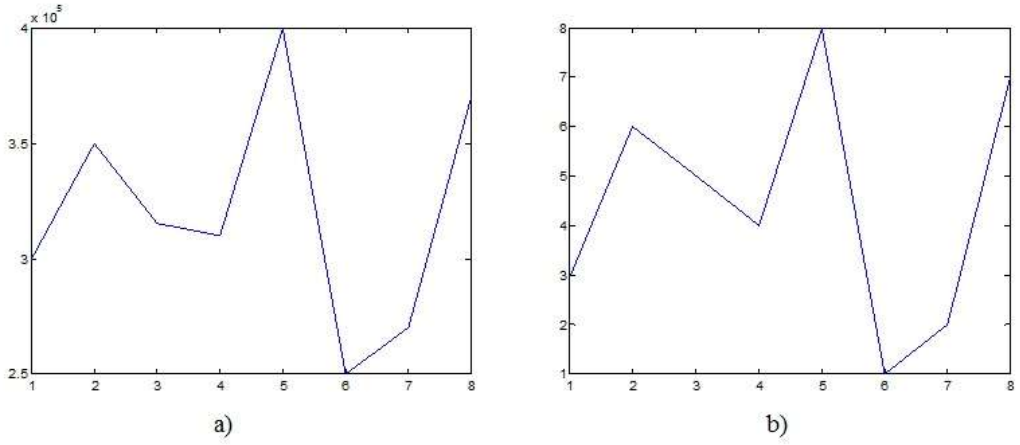


Figure 20 – New transformation method illustration. An input vector (a) of any dynamic range is transformed in a new vector (b) with integer values from 1 to 8, which is the vector length, in function of the relative value of the components.

Then, the comparison method can be described in the following steps:

1. For each studied method, two vectors of 2000 components must be compared. The first one contains the values of the image files coded in JPEG format. The second one contains the results of applying the specific method to each image. As explained in Chapter 3, the values are the mean of the resulting image after applying the corresponding filter.
2. Since the two vectors have very different dynamic ranges, the previous described transformation is applied. After that, there are two new vectors with values from 1 to 2000.
3. The difference of the two vectors is calculated component by component, and the mean of the new differences vector is computed. This will be the value assigned to the corresponding method. The lower that value is, the higher the correlation between the two original vectors will be.

Moreover, in the case of MATLAB edge detection function, an additional process is performed, since there is the option to select a detecting threshold for the found edges. For this reason, this parameter is optimized by applying the previously described procedure using a large number of values for this threshold, as described in Figure 21.

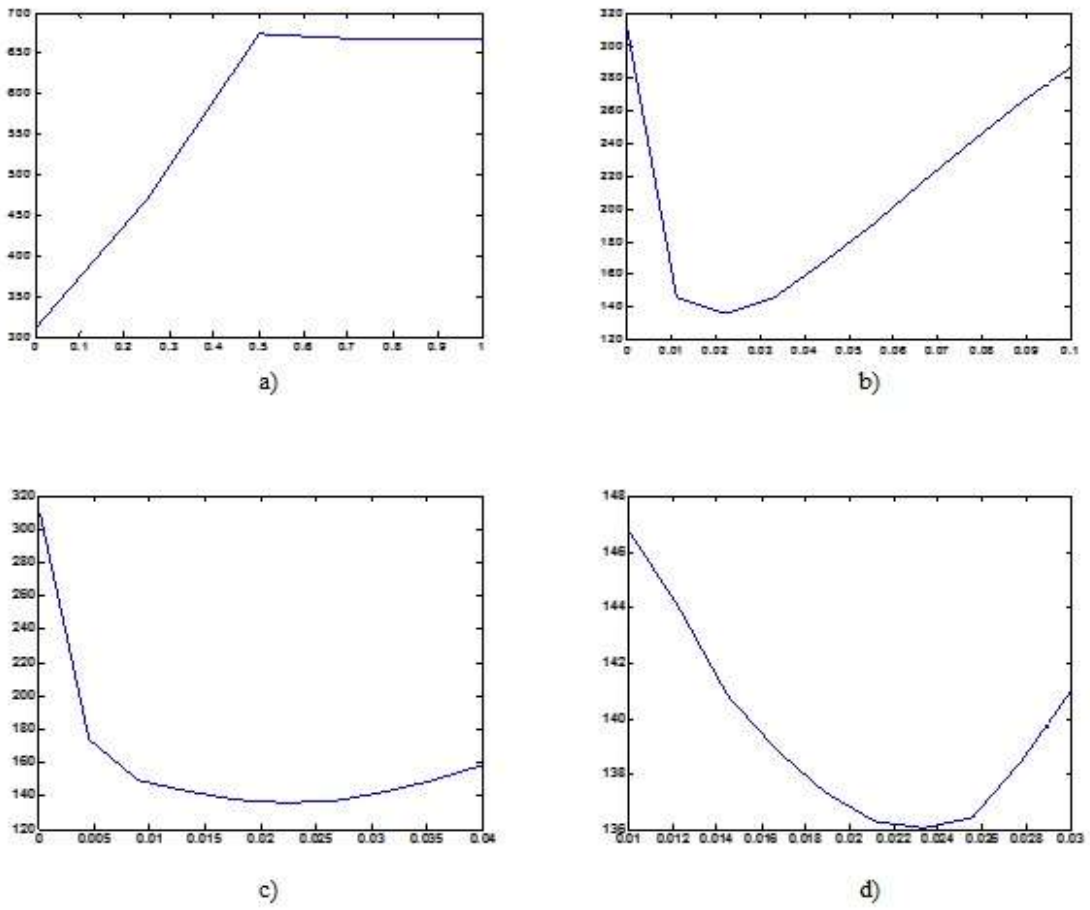


Figure 21 – Optimization of the detection threshold for ML Sobel. Many values are tested within the ranges from 0-1 (a) to 0.01-0.03 (d).

4.2.2. Results

Once the evaluation method is described, it is time to see the obtained results, in Table 2 and Figure 22.

METHOD	RESULT
Sobel	95.9
Prewitt	111.9
FT	509.1
ML Sobel	136.1
ML Prewitt	136.1
ML Roberts	139.3
ML LOG	131.1
ML Canny	212.2

Table 2 - Results

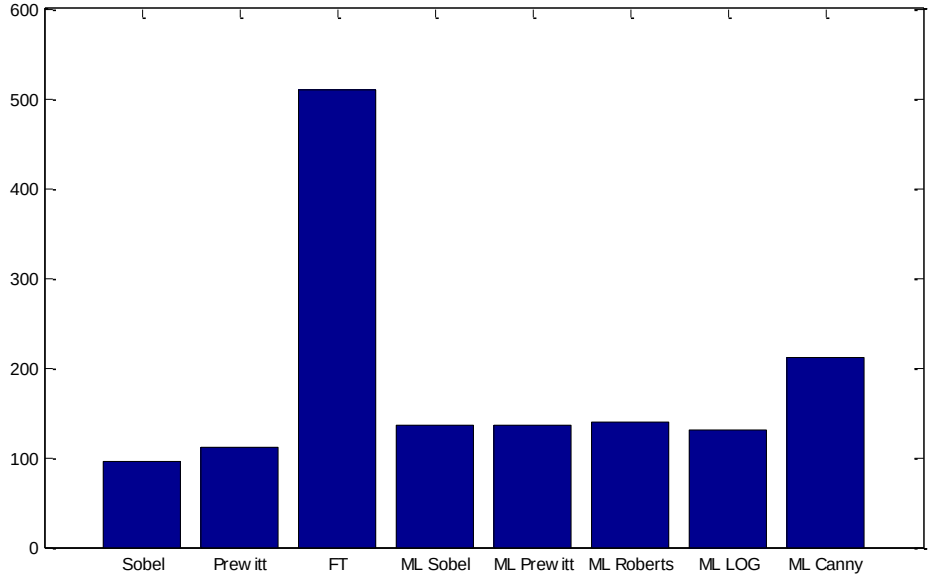


Figure 22 – Results plot

The best result is obtained using a Sobel filter. This is quite a good fact for the project development, since applying a simple 3x3 filter to the images is a reasonably easy task, in terms of computing cost. Prewitt filter and MATLAB edge detection function provide slightly worse results, while FT based method result is clearly worse.

4.4. Global complexity

4.3.1. Introduction

Once the edges estimation has been done, four vectors are available to work with in order to find a global complexity estimation single vector, as it was explained in Chapter 3. They are Edge Information (EI), Motion Information (MI), the product EIxMI and the derivative of EI, EI'. By combining the value of all four vectors, a good estimation of the video complexity must be found. To find the best way to do it, the obtained references before described have been used to test many linear combinations of the four vectors.

In addition, there are two weighting vectors which must also be optimized. As explained in Chapter 3 in the section of image extraction, different possibilities were possible, regarding the number of extracted images at each point and their distribution.

Finally, the following distribution was decided to be applied for the studied points at Frame N: [Frame N, Frame N+1, Frame N+2, Frame N+5, Frame N+10, Frame N+20]. It has been chosen because it combines a reasonably reduced number of extracted frames per studied point while it provides a representation of the information of almost a second of the video. Then, the two described methods for edge and motion information detection are applied to all images. Thereby, there are six values of Edge Information and 15 for Motion Information (all the possible subtractions, two by two) at each studied point. However, a single value must be obtained for both parameters and thus all the values must be averaged. To do it, different weighting matrices have been tested, as it is shown in Figures 23 and 24. This means that, apart from testing many linear combinations of the four available methods, the different weighting vectors for EI and MI have to be tested for each combination too, since EI and MI vectors are constructed by averaging all the values at each point.

Available values:

E_extracted(n) = [E(n), E(n+1), E(n+2), E(n+5), E(n+10), E(n+20)]

Weighting vectors:

W_EI_1 = [1, 0, 0, 0, 0, 0]
W_EI_2 = [1, 1, 1, 1, 1, 1]
W_EI_3 = [6, 5, 4, 3, 2, 1]
W_EI_4 = [30, 20, 10, 5, 2, 1]

EI(n) = E_extracted(n)*W_EI_I

Where W_EI_I is the weighting vector providing best result.

Figure 23 – Edge information weighting procedure

Available values:

M_extracted(n)= [M(n,n+1), M(n,n+2), M(n,n+5), M(n,n+10), M(n,n+20),
M(n+1,n+2), M(n+1,n+5), M(n+1,n+10), M(n+1,n+20),
M(n+2,n+5), M(n+2,n+10), M(n+2,n+20),
M(n+5,n+10), M(n+5,n+20),
M(n+10,n+20)]

Weighting vectors:

W_MI_1 = [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
W_MI_2 = [1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
W_MI_3 = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
W_MI_4 = [5, 5, 5, 5, 5, 4, 4, 4, 4, 3, 3, 3, 2, 2, 1]
W_MI_5 = [20, 20, 20, 20, 20, 10, 10, 10, 10, 5, 5, 5, 2, 2, 1]

MI(n) = M_extracted(n)*W_MI_I

Where W_MI_I is the weighting vector providing best result.

Figure 24 - Motion information weighting procedure

4.3.2. Evaluation method

In total 131760 combinations have been tested and evaluated for each one of the ten reference videos, between weighting vectors of EI and MI and linear combinations of the four available vectors. To find the best combination, two measures were calculated. The first one was the accuracy in detecting complex zones. It is just the rate of the well found zones from the ones labelled in the references, as it is explained in Figure 25. However, different combinations could produce the same result, and it gives no information about the order of complexity among the complex zones. For this reason, a second measure was added. It takes into account the value of the estimated vector at the exactly moment were a complex part is marked at the reference. Then, the relative difference of the values between the reference and the estimation is measured in exactly the same way as with the edges estimations: a value is given from 1 to N and the differences are calculated component by component. It is illustrated in Figure 26. Then, the mean of the resulting subtracting vector is calculated. By using both two measures, the best combination can be found from all the calculated ones.

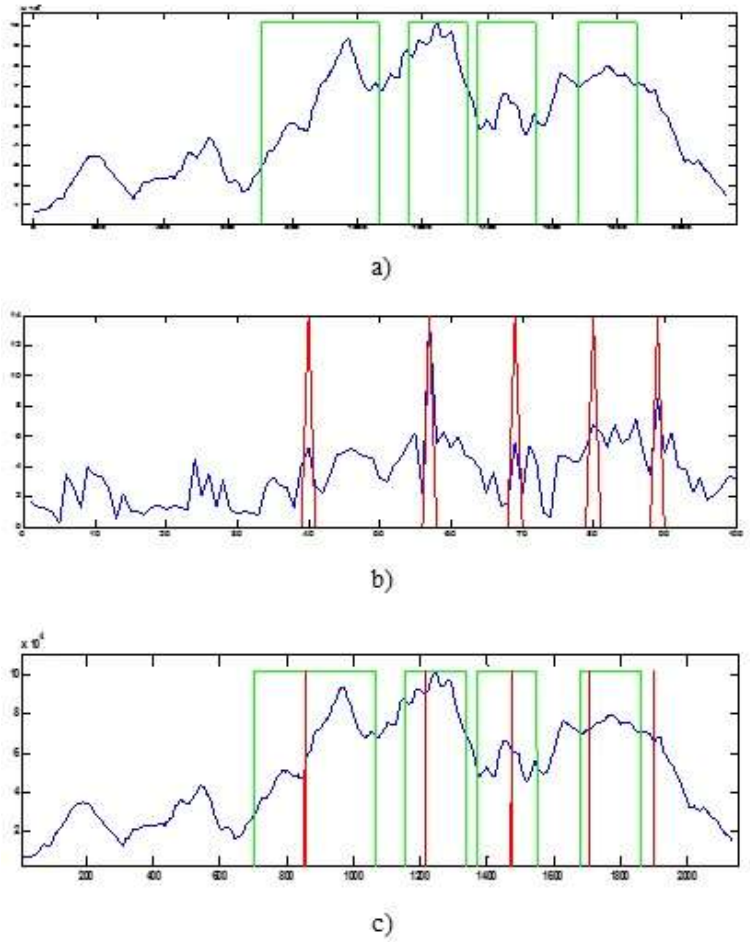


Figure 25 – Accuracy measuring. a) The reference with the five most complex zones marked in green. b) The obtained complexity estimation with the found five most complex zones. c) Both signals are compared in order to calculate the accuracy. In this case, 4 out of 5 complex zones were found.

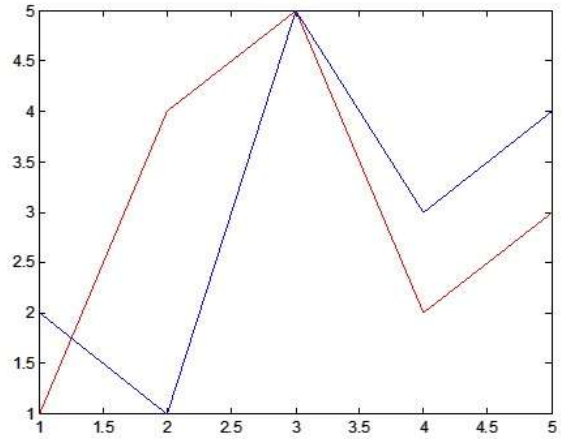


Figure 26 – Relative difference of the most complex zones. In red, the order of the five found zones in the reference. In blue, the order of the values of the complexity estimation vector in the same instants (note that they can be different from the five most complex found zones)

Some other parameters have also been optimized in this process. They are related to the way in which the signals are processed in order to detect the most complex

zones. For example, a low-pass filter is applied to the obtained signal and then a MATLAB function is used to find the vector peaks. This function includes a parameter to set the margin in which two different local peaks will be considered part of the same peak. The filter width and the margin value have been manually optimized by a trial and error procedure, to ensure a good behaviour in all kind of videos.

Besides that, the width of the labelled zones in the references is an important point to take into account when the accuracy is calculated. That width has been chosen in order to ensure that a point will be considered well found only if it is close enough to the peak.

4.3.3. Results

After testing all the combinations, each one of them has a value of accuracy and another of relative difference associated, as it can be seen in Figure 27.

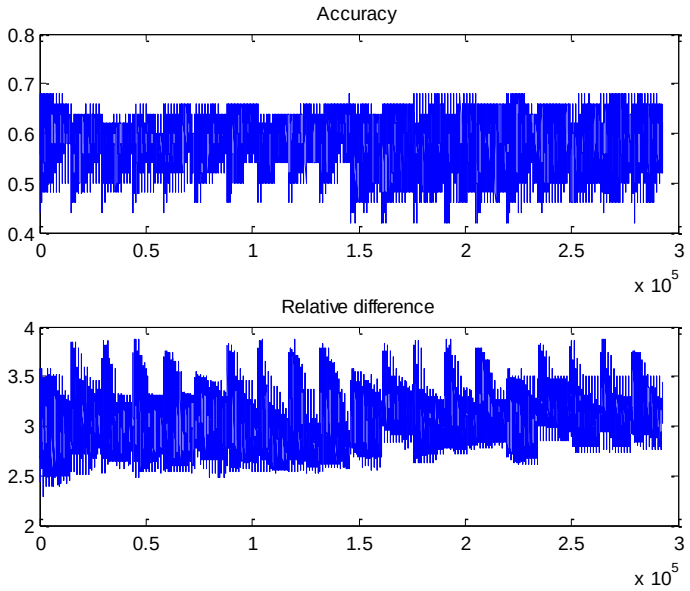


Figure 27 – A first representation of all combinations results

In Figure 28, two different kinds of ordering the combinations are shown: by the accuracy and by the relative difference. It is clear that the dynamic range of the values of the non-sorted parameter of the best combinations in the sorted parameter is too large to easily find a good combination for both parameters.

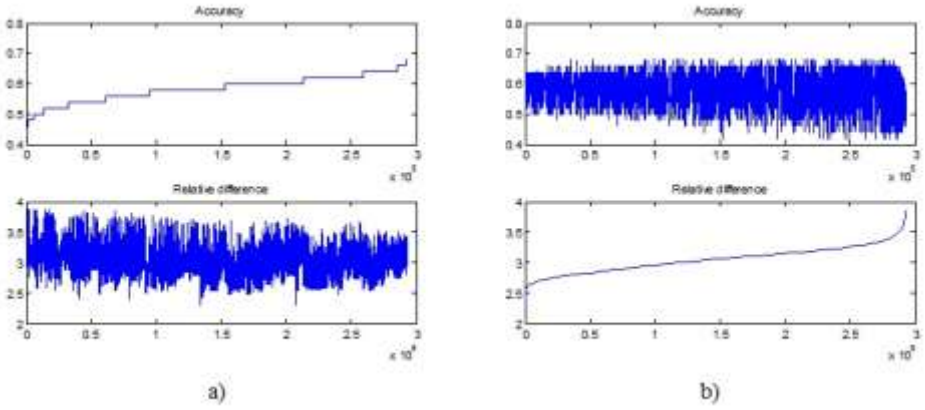


Figure 28 – All combinations sorted by accuracy (a) and relative difference (b).

For this reason a two-step optimizing process has been followed. First of all, the combinations with best accuracy have been selected, without taking into account the relative difference of the peaks. It makes sense since the accuracy is the most desired parameter to be optimized. Then, all that good combinations are sorted by the relative difference, as it is shown in Figure 29. Finally, only the ten best combinations are selected. This ensures that the final studied combinations provide a good result firstly in accuracy and also in relative difference of the peaks.

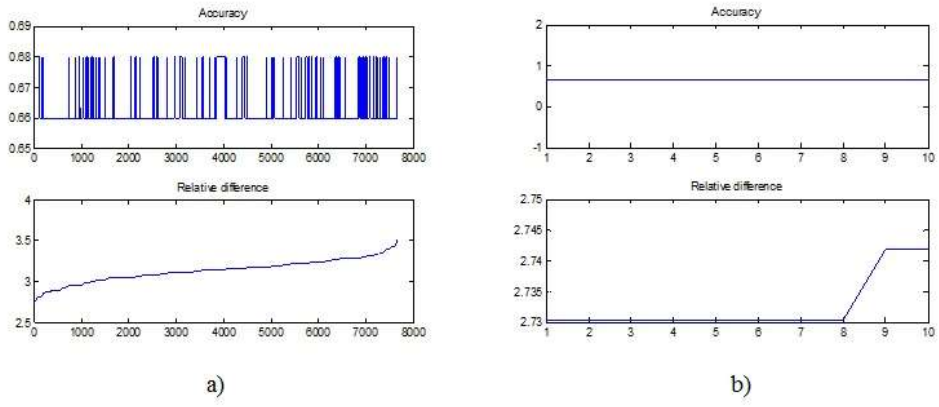


Figure 29 – a) The combinations with best accuracy, sorted by the relative difference. b) The ten overall best combinations.

Once the best combinations are found, it is time to study them. As it can be observed in Figure 30, the first eight combinations provide exactly the same values of accuracy and relative difference. For this reason, they will be the only ones taken into account.

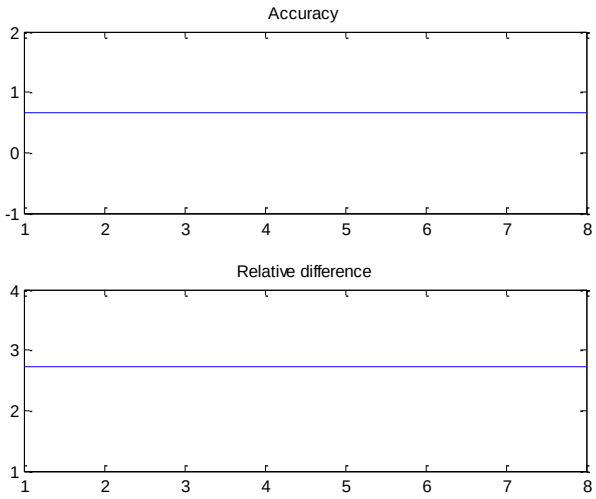


Figure 30 – Final results of the best eight combinations

In Table 3, the descriptions of these combinations are shown.

N	W_EI_I	W_MI_I	Linear combination
1	4	1	[3 5 5 5]
2	4	1	[3 5 6 5]
3	4	1	[3 8 7 7]
4	4	1	[5 8 8 8]
5	4	1	[3 8 9 8]
6	4	1	[6 10 9 9]
7	4	1	[6 10 10 9]
8	4	1	[6 10 10 10]

Table 3 - Weighting vectors and linear combinations that provide the best results

It seems clear that the 4th edge weighting vector and the first motion weighting vectors are the best ones. However, it is difficult to decide which linear combination should be selected. In Figure 31, all the combinations haven been normalized and represented together (they have been actually normalized during the test). It can be seen that the relative values of the vector components are very close and, since calculating the mean of all vectors will not ensure a best performance, combination number two has been arbitrarily selected.

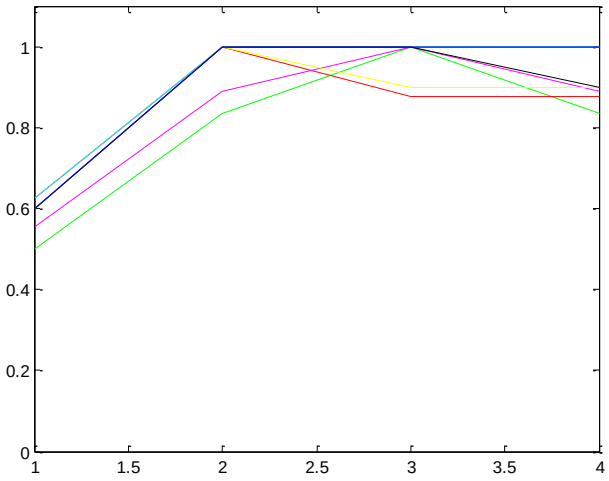


Figure 31 – Representation of the normalized 8 best combinations. They are all very similar.

With these results, the optimum parameters can be fixed. For the edge estimation, the best weighting matrix is [30, 20, 10, 5, 2, 1], which calculates the value taking into account all six images, but giving more importance to the ones which are close to the studied point. In the case of the motion information, [1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0] vector is selected. It means that it only uses the subtraction between the actual frame of the studied point and the immediately following one to create the estimation. Finally, EI, MI, EIxMI and EI' vectors will be combined in the following way: [3 5 6 5].

4.5. Full system

Although the final integration of all the functions in the Graphical User Interface will be explained in next Chapter, a first evaluation of the system behaviour can be previously done.

4.4.1. Time evaluation

The needed time to complete a full transcoding process highly depends on the selected video and on the used processor specifications. However it can be compared with the needed time if the process must be done with a trial and error procedure, as improving this time is one of the main goals of the project.

Firstly, this trial and error time will be calculated, testing the process with a short video which lasts 143 seconds. A full transcodification of this sequence will take 4 time units (expressing it in seconds is meaningless since it depends on the processor, as explained). In one of the best assumptions, the user will need at least three trials to get a good final solution. For example, with the two first attempts the user could find two boundaries of too much and insufficient quality, while in a good choice the third one

could fulfil the requirements being placed between the two previous ones. This means that 12 time units will be needed to complete the process.

With the implemented application, a first step of complexity estimation will be needed. This estimation will last 5 time unites with the same video. However, several previews can then be quickly generated. For example, creating 8 different previews with different parameters will last 1 time unit. Finally, the whole video must be transcoded only once, adding 4 time units to the process.

As it can be seen in Figure 32, the traditional procedure would last 12 time units in total, while by using this application the process can be done in only 10 time units. In addition, the larger the videos are, the higher the time to transcode them is, while the estimation process time can be almost the same as before. For this reason, the difference between the two methods will be increased if the video duration is also increased. Finally, it has to be said that although these presented values have been empirically measured, it is hard to establish a time relation between the two methods, since it also depends on the user specifications when the video has to be transcoded, apart from the video duration and the processor specifications.

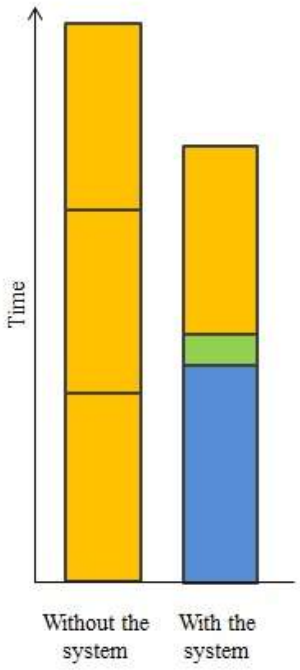


Figure 32 – Time evaluation

4.4.2. Accuracy evaluation

As the result of the evaluation process described in section 4.3, the accuracy of the complexity estimation can be fixed at 66% of found complex zones. This value corresponds to the following evaluation method: placing a 6 second window around a marked complex zone in the reference, a point will be considered as well detected only if a 6 second window placed around the found point in the estimation contains, at least, 2/3 of the first window. It is obvious that if this restriction is reduced, the accuracy will

be increased. Actually, if 1/2 of covered window is considered enough (which will mean that the detected window includes the marked point), a 74% of accuracy is reached. However, the first restriction was used for practical reasons in the evaluation process.

4.6. Fast version implementation

As the complexity estimation process is clearly the bottleneck of this project, an attempt to reduce it has been studied. This approach is based on reducing the number of extracted frames while trying to keep the accuracy as good as possible. This process was partially done when the weighting vectors where optimized, since some options reduced the importance of the last extracted frames. However, in this case a single situation is evaluated: using only one extracted frame at each studied point (in fact two consecutive frames must be extracted every time for the motion estimation).
The previously evaluation process has been performed again for this case, with the difference that it makes no sense to optimize the weighting vectors, since it has been decided to use only one frame per studied point (two for the motion estimation).

With this fast version, user will be able to speed the process in the cases in which there is a lower accuracy requirement in selecting the actual most complex zones.

4.5.1. Time evaluation

Regarding the time needed, the estimation process has been reduced about 80%. This leads to a higher time difference from the trial and error method, as it can be seen in Figure 33.

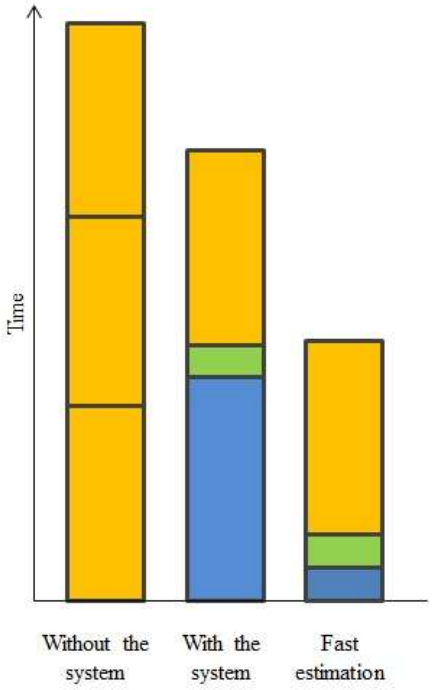


Figure 33 – Time evaluation including fast estimation

4.5.2. Accuracy evaluation

The obtained accuracy after the evaluation test for the fast implementation is 54% of found complex zones. However, as explained in the full implementation case, if the window restriction reduced at 50%, the accuracy is increased. In this case, it reaches 68%.

Chapter 5

SYSTEM GRAPHICAL USER INTERFACE

In this chapter, the implementation and the functionalities of the final developed GUI will be described. First of all, there will be a review of the available options and the program behaviour and, finally, the implementation of the different parts will be described.

5.1. GUI description

To describe the final design of the GUI, all user possibilities will be reviewed as a virtual guide to the application.

To begin with, the user can name the current project and has to select an input video to be studied. Once the video is selected, its information appears at the top of the window: file size, duration, frame rate, codecs, etc. At the left part, the user can introduce the desired video parameters. These parameters may not be the final ones, since the goal of the application is to find them, but they can suppose a first step for the application to begin with. Anyway, the user can use the application as a simple transcoder by selecting there the coding parameters and ask for a simple transcodification. These steps are illustrated in Figure 34.

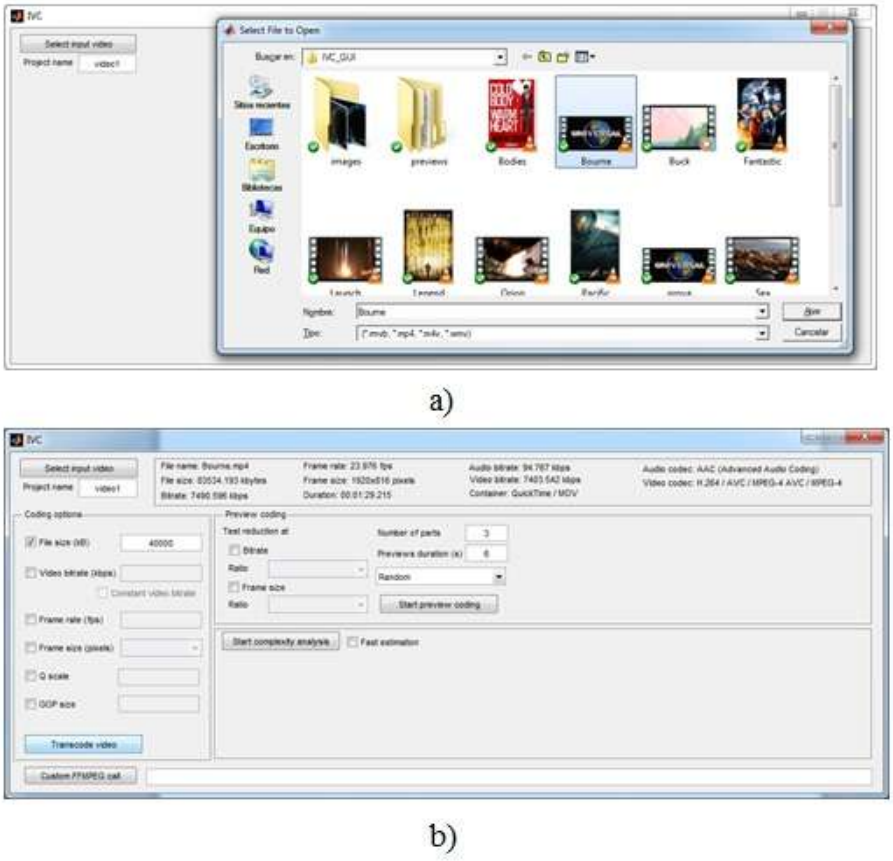


Figure 34 – Running the application. a) Input video selection. b) Once the video is selected, the information is shown and user can introduce coding parameters.

Then, the user can start with the preview coding study. At this part, the user can ask the program to transcode as many parts of the video as desired, selecting also the duration of them. Then, bitrate and frame rate reductions can be tested, by selecting from the menus some reduction ratios. This will apply to all the asked transcodings and thus the number of new generated sequences will be increased. At this moment, random parts of the video may only be selected to apply the transcodifications just there, since no more information of the video is available. However, this is only a fast possibility for the user, while the more interesting execution will include a complexity estimation of the video, to apply all transcodings in the most difficult to code parts. Actually, if a fast execution is wanted, the fast complexity estimation option can also be selected. All this can be seen in Figure 35.

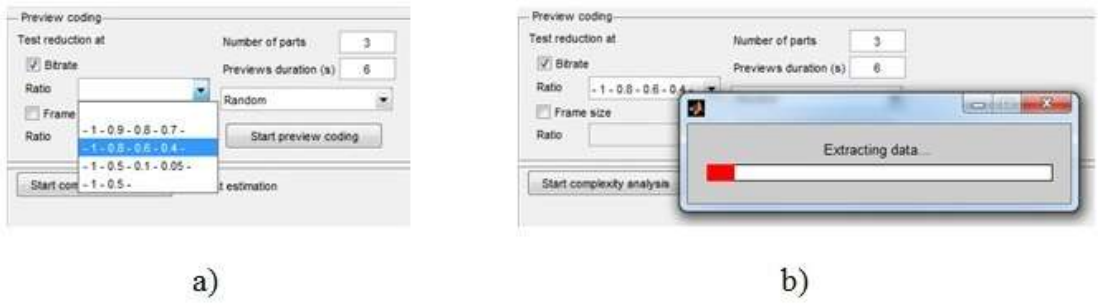


Figure 35 – a) Bitrate and frame rate reductions can be tested using the GUI menus. b) To know where the video most complex parts are, the user can start the complexity analysis.

When the analysis is finished, a complexity vector appears at the bottom part of the window, with the most complex parts marked. Apart from the global complexity vector, all previously calculated complexity measures can be visualized in the same area: edge information, motion information, their product and the derivative of the edge information. With all this information, user can exactly select where to apply the transcodifications. As it can be seen in Figure 36, besides using random parts, the most complex parts can be selected and there is also the freedom to manually select any part of the video that user think it can be interesting.

Apart from visualising all vectors, user can directly extract actual and processed frames along the video, as it is shown in Figure 37.

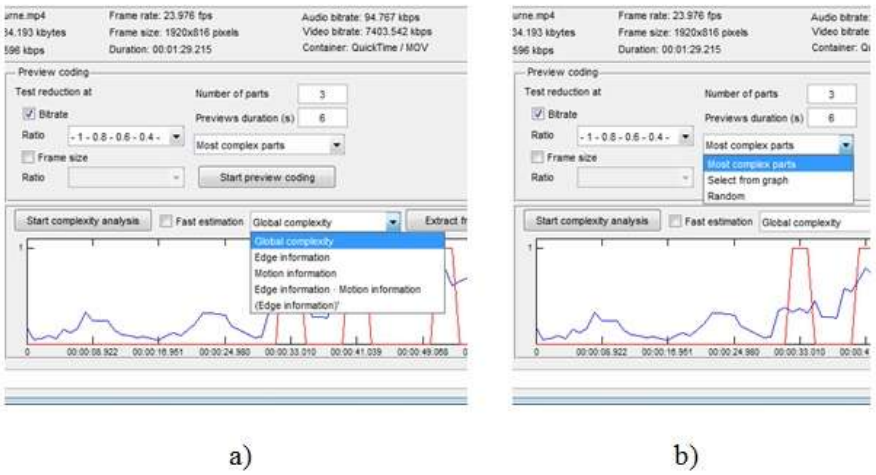


Figure 36 – a) All information vectors can be plotted in the complexity analysis part. b) User can select where to apply the transcodifications.

a) b) c)

Figure 37 – Any frame can be extracted and visualised a) as itself, b) by its edge information and c) by its motion information.

When the desired parts to be transcoded are decided and the process is finished, video files are stored in the program directory. At the same moment, they can be seen in the preview coding part, as a list where each video file has its information about time, bitrate, frame rate and extrapolated whole video size, as it can be seen in Figure 38. From this menu, user can select which sequence to visualize. In addition, every time a transcodification is performed, its FFMPEG command is presented in the very bottom of the screen. This allows the user to understand the process and to perform a manually generated transcodification.

In Figure 39, two frames from the same time are presented. The first one is extracted from a sequence coded with a higher bitrate and thus the video quality is clearly better. By visualizing the generated previews, the user can decide which coding parameters are better to be used. Moreover, if any of them fulfil his or her requirements, all the parameters can be changed and all the preview process can be repeated again, without needing a new complexity estimation, since it will be still available.

Actually, the GUI implementation is thought to allow this kind of execution: letting the user interact with the application and ask for as many previews as desired, in function of the information provided by the program.

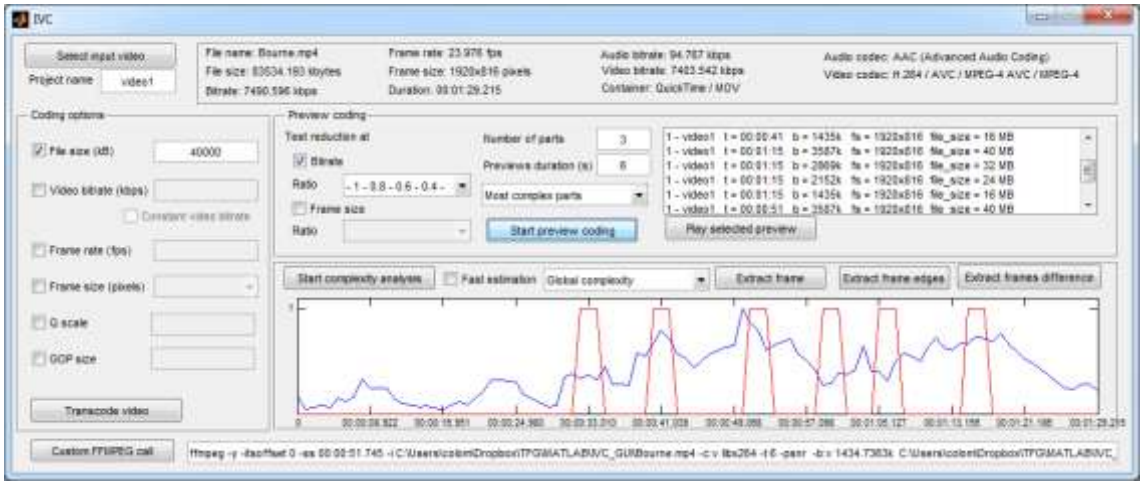


Figure 38 – Once the previews are generated, they can be visualized using the new appeared list in the preview coding part.

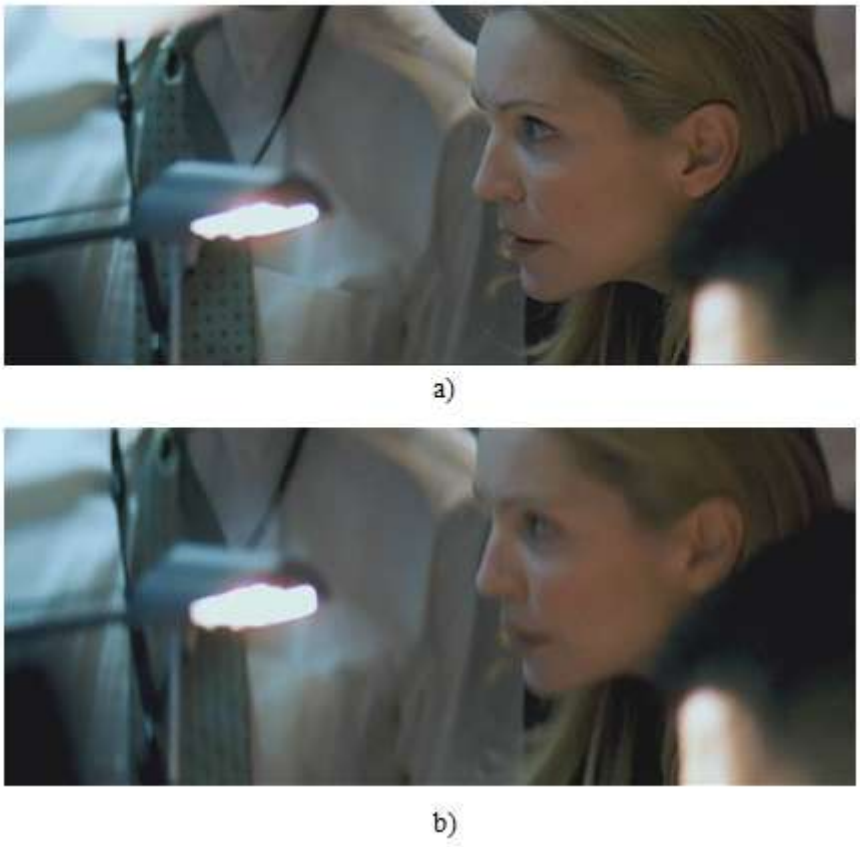


Figure 39 – Two frame from the generated previews. Is by visualizing the result of choosing different parameters that user can find which are the best coding parameters.

5.2. GUI implementation and functions integration

Three main parts of the final GUI will be explained, in order to describe the process in which all functions have been integrated.

5.2.1. Simple transcoder

At the left part of the GUI, there is a simple transcoder, where the user can introduce the desired coding parameters and use them to ask for a simple transcodification. Moreover, these parameters will also be the ones that the preview coding part will use for its study.

When the user press the “Transcode video” button, the program gets all the needed information to then call FFMPEG. Input video information is already available since the file has been selected, while the parameters introduced by the user are processed and stored to be used by other functions. If any of the possible input parameters are not specified, the transcodification will be performed keeping them from the input file.

After getting all the information, the application has to format it in order to fulfil FFMPEG syntax. For this reason, some characters are added to the introduced

parameters and some conditional statements are implemented to avoid possible conflicts between coding parameters. For example, video file and bitrate cannot be introduced at the same time, since they depend on each other.

When all parameters are formatted, a single string command is constructed and it is used to call FFMPEG. The previous processing ensures that everything is correct and FFMPEG will execute without problems. The process finalise when a new video file is created.

5.2.2. Complexity estimation

To begin with the complexity estimation, all frames must be extracted along the video, by using FFMPEG calls as it was explained in Chapter 3. They are stored in a specific directory for the current project. Then, all frames are read and processed one by one and their information is stored in MATLAB variables.

This process is explained in Chapter 4, like the next step. When all the information is stored, it has to be processed again, in order to obtain the four information vectors. Finally, they are linearly combined and a single complexity estimation vector is found.

If a fast estimation is selected, the process is slightly different. There are less extracted frames (only two consecutive ones per studied point) and therefore a single processing step is needed to extract the information vectors from the extracted frames. Then, the final complexity vector is calculated using a linear combination which is different from the one used in the normal estimation.

Finally, this vector has to be displayed to the user, with the addition of the most complex zones marked over it. This labelled time intervals are not only a visual clarification but also an important parameter to be used in the preview coding part. Besides this, a menu is also implemented to display any of the vectors which are available in the running application: edge information, motion information, etc. The selected one is automatically presented at the same axis as the first one. Moreover, FFMPEG calls are used to let the user to extract actual o processed frames, just selecting the instant by clicking to the graph.

5.2.3. Preview coding

To handle the preview coding process some information from other functions is needed. First of all, the input file information is required to construct an FFMPEG call. Then, as it was explained, the introduced values in the simple transcoding part are also used to base on them the bitrate or frame rate reduction tests. And finally, the most complex parts calculated in the complexity estimation function are used to place there the transcodifications if it is desired by the user.

However, they can also be placed in a random part of the video or in a specific moment selected by the user. This is all the information needed to start creating several single commands, which will be used to call FFMPEG. There must be one command for

each one of the possible combinations between selected moments, bit rate reduction and frame rate reduction. Once generated, they are executed one by one, creating new video files which will be stored in a new directory.

With all the parameters used to make the commands, a new list of the generated previews appears at the screen, specifying these parameters in order to let the user identify each one. By selecting the elements of the list, user can visualise all the previews. This is how all the process is finalized and initialized again: watching the video transcoded with different parameters lets the user choose the best ones or keep looking for them by starting again the process.

Chapter 6

CONCLUSIONS

6.1. Overall evaluation

This application was designed with the main aim of improving the video transcoding process. Taking into account the classical solution based on a trial and error procedure, a new model had to be implemented. After reviewing the implementation process and the program behaviour, it can be said that it really provides a new good solution for the studied problem.

That is also true in terms of needed time to complete the process. It can be said that this implemented solution clearly overcomes the classical one by speeding up the whole process, as it has been explained in the previous Chapter. Moreover, besides the reduction of the needed time, the application provides a useful tool with an important value by itself. It helps the user to get into the transcoding process, to identify why some solutions are not good enough and to decide what to do to find a good one. Even if the needed time was not reduced enough, the practical options provided to the user would make it a success anyway.

6.2. Goal accomplishment

As explained in other chapters, the proposed solution is based on the three main goals that were defined at the beginning. For this reason, it is meaningful to review them:

- Preview coding. The tool that allows the user to ask for several short previews has been successfully implemented. User can select which parts to transcode and which coding parameters to test and the application generates all the corresponding previews with the specified duration. Then, user can easily visualize them, and thereby evaluate the results.
- Complexity estimation. Since the preview part needs the video most complex parts to reach its optimum performance, having found them has also been an important step for the whole application development. This part has been also successfully implemented, as it was described in Chapter 4.
- Interactivity. The third main goal was not developing a single part by itself, but it is related to how the user can execute the application and work with it. As it was planned, there is a high level of freedom for the user to ask for complexity estimations and previews and to decide at every moment which way to follow to

find the optimum parameters, after watching at the program results after each step.

6.3. Possible further work

Finally, although the developed application totally fulfils the initial requirements, some aspects might be improved or deeply studied can be commented, in order to increase the program accuracy or to adapt it to other situations.

First of all, the processing part of the extracted frames is one of the deep study focuses. Besides all edge detection evaluated techniques, some others could also be tested, especially the ones related to the frequency domain, which is actually the one used in the blocks transformation and compression. However, the finally chosen technique provides a very good result in the complexity estimation, as it have been proved in Chapter 4. In addition, some other methods could be studied for the motion estimation, like calculating motion vectors between frames, with the hard trade-off of increasing the processing complexity and thus the execution time.

Another aspect that can be deeply studied is the distribution of the studied points in the complexity estimation. Since the development was thought to work with short videos (up to fifteen minutes duration, approximately), the introduction of full movies or large videos would not produce as good results as with shorter ones. Obviously, all the system will be correctly executed, but the accuracy in the complexity estimation will not be good enough to ensure that all complex parts were found. For this reason, adapting the frame extraction to make it able to work with large videos would be an interesting focus in further work.

Chapter 7

BIBLIOGRAPHY AND REFERENCES

[1] Kabeen, K. and Gent, P., *Image compression and the Discrete Cosine Transform*, Math 45, College of the Redwoods.

[2] Eklund, A., *Image coding with H.264 I-frames*, Linköpings Universitet, 2007.

[3] Richardson, I., *H.264 / AVC Intra prediction*, Vcodex.com, 2011.

[4] Richardson, I., *H.264 / AVC Inter prediction*, Vcodex.com, 2011.

[5] Richardson, I., *4x4 Transform and Quantization in H.264/AVC*, Vcodex.com, 2010.

[6] Wiegand, T., Sullivan, G., Bjøntegaard, G. and Luthra, A., *Overview of the H.264-AVC Video Coding Standard*, IEEE Transactions on Circuits and Systems for Video Technology, vol. 13, no. 7, 2003.

[7] Richardson, I., *H.264 / AVC Context Adaptive Binary Arithmetic Coding (CABAC)*, Vcodex.com, 2010.

[8] Richardson, I., *H.264 / AVC Context Adaptive Variable Length Coding*, Vcodex.com, 2010.

[9] Webster, A. and Wolf, S., *Subjective and Objective Measures of Scene Criticality*, Institute for Telecommunication Sciences (NTIA/ITS), 1997.

[10] Wu, H., Claypool, M. and Kinicki, R., *A Study of Video Motion and Scene Complexity*, Worcester Polytechnic Institute.