MASTER THESIS

TITLE: A Real-Time Multiple Description System

MASTER DEGREE: Master of Science in Telecommunication Engineering & Management - MASTEAM

AUTHOR: Manuel Bermejo Plana

DIRECTOR: Sergio Machado Sánchez

DATE: January 14, 2010
Overview

In this thesis, we implement and analyze a Multiple Description Coding (MDC) system for video streams captured from a live video input in real-time. The input source is a web-cam over Linux. Besides, the algorithm originates four descriptions from the original video stream that could be decoded independently. Each description is part of the entire image downsampled in space domain. Every description is compressed independently with MPEG-4 part 2 or H.264 and high or low bitrate profile. Video is sent over a communications network. In case of any error at any description, including the complete lost of a subset of them, the system can deal with the problem and reconstruct the entire video stream, with a loss of quality that is analyzed in terms of the Peak Signal-to-Noise Ratio (PSNR) objective quality parameter. We study different algorithms to the reconstruction and post processing of the entire sequence in order to minimize the visual effects related to MDC. Furthermore, the quality at the receptor is analyzed for any subset of received descriptions.
# CONTENTS

## INTRODUCTION ...................................................... 1

## CHAPTER 1. MDC System Architecture .......................... 3
  1.1. Overview .................................................. 3
  1.2. General System Architecture .......................... 4
    1.2.1. Capture and coding system ......................... 4
    1.2.2. Receiver and decoding system ................... 8

## CHAPTER 2. Video Coding System Description .................. 11
  2.1. Gamma correction ...................................... 11
  2.2. The MPEG-4 and H.264 Standards ....................... 12

## CHAPTER 3. Recovering from description lost .................. 15
  3.1. Digital image interpolation ............................. 16
    3.1.1. Near Neighbour Replication (NNR) ............... 16
    3.1.2. Bilinear ............................................. 17
    3.1.3. Kernel implementation .......................... 17
  3.2. Edge Sensing ........................................... 17
  3.3. Variable Number of Gradients .......................... 18
  3.4. Post filtering .......................................... 19
    3.4.1. 2D filter .......................................... 19
    3.4.2. Low-pass filter ................................. 20

## CHAPTER 4. MDC System Implementation ......................... 21
  4.1. Capture and coder classes .............................. 21
    4.1.1. YUVFrame ........................................... 21
    4.1.2. YUVReader ......................................... 21
    4.1.3. Subsampler ......................................... 23
    4.1.4. Server ............................................. 24
    4.1.5. Coder ............................................... 24
LIST OF FIGURES

1.1 General architecture .................................................. 4
1.2 Acquisition block system .............................................. 4
1.3 Subsampler block system .............................................. 5
1.4 Chroma subsampling patterns ......................................... 5
1.5 Multiple Description Coding ......................................... 7
1.6 Reception block system ............................................... 8
1.7 Inverse Multiple Description Coding ................................. 9
2.1 Gamma correction ..................................................... 12
2.2 MPEG-4 part 2 object-based coding .................................. 13
2.3 Comparison between MPEG-2, H.264, MPEG-4 part 2 and H.264 . 14
3.1 Luminance loss pattern for a) one description lost, b) two descriptions lost, c) three descriptions lost ................................................... 15
3.2 NNR vs. Bilinear .......................................................... 16
4.1 YUVFrame class diagram ............................................... 21
4.2 YUVReader and its inherited class diagram .......................... 22
4.3 YUVURLReader class diagram .......................................... 22
4.4 YUVFileReader class diagram .......................................... 22
4.5 YUVRTPReader class diagram .......................................... 23
4.6 YUVUDPReader class diagram .......................................... 23
4.7 Subsampler class diagram .............................................. 24
4.8 Server class diagram .................................................... 24
4.9 Subsampler class diagram .............................................. 24
4.10 Receiver class diagram ................................................ 25
4.11 Decoder class diagram ................................................ 25
4.12 SubsamplerInv class diagram ......................................... 26
5.1 Original video stream encoded and decoded ......................... 27
5.2 MDC with description lost ............................................. 28
5.3 Y-PSNR for any description lost ...................................... 29
5.4 Y-PSNR for any description lost at 256kbps ......................... 30
5.5 Y-PSNR for any description lost at 512kbps ......................... 30
5.6 Original and difference between frame 205 ......................... 31
5.7 Y-PSNR MPEG-4 part 2 - One description lost at 256kbps/description ................................................... 32
5.8 Y-PSNR MPEG-4 part 2 - One description lost at 256kbps/description. ES and VNG .................................................. 32
5.9 Y-PSNR MPEG-4 part 2 - One description lost at 256kbps/description. ES with and without post filtering ................ 33
5.10 Y-PSNR MPEG-4 part 2 - One description lost at 256kbps/description. Post filtering .................................................. 34
5.11 Y-PSNR MPEG-4 part 2 - One description lost at 256kbps/description. Difference between ES and VNG with post filtering ................ 34
5.12 Y-PSNR for one description lost - ES with post filtering ........ 35
5.13 Y-PSNR for one description lost - ES with post filtering at 256kbps/description . 36
5.14 Original and difference between frame 2 . . . . . . . . . . . . . . . . . . . . . . 36
5.15 Y-PSNR for one description lost - ES with post filtering at 512kbps/description . 37
5.16 Y-PSNR MPEG-4 part 2 - Two descriptions lost at 256kbps/description . . . . 38
5.17 Y-PSNR MPEG-4 part 2 - Two descriptions lost at 512kbps/description . . . . 39
5.18 Y-PSNR x264 - Two descriptions lost at 256kbps/description . . . . . . . . . 39
5.19 Y-PSNR x264 - Two descriptions lost at 512kbps/description . . . . . . . . . 40
5.20 Y-PSNR for two descriptions lost - Bilinear interpolation . . . . . . . . . . 40
5.21 Y-PSNR for two descriptions lost at 512kbps/description - Bilinear interpolation.
Difference between x264 and MPEG-4 part 2 . . . . . . . . . . . . . . . . . . . . . . 41
5.22 Y-PSNR for two descriptions lost at 256kbps/description - Bilinear interpolation.
Difference between x264 and MPEG-4 part 2 . . . . . . . . . . . . . . . . . . . . . . 42
5.23 Y-PSNR MPEG-4 part 2 - Three descriptions lost at 256kbps/description . . . 43
5.24 Foreman video stream. High moving sequence . . . . . . . . . . . . . . . . . . 44
5.25 Y-PSNR MPEG-4 part 2 - Three descriptions lost at 512kbps/description . . . 45
5.26 Y-PSNR x264 - Three descriptions lost at 256kbps/description . . . . . . . . 45
5.27 Y-PSNR x264 - Three descriptions lost at 512kbps/description . . . . . . . . 46
LIST OF TABLES

2.1 Summary of differences between MPEG-4 part 2 and H.264 ............... 14

5.1 Average Y-PSNR (in dB) for any description lost ..................... 29
5.2 Average Y-PSNR (in dB) for 1 description lost ......................... 37
5.3 Average Y-PSNR (in dB) for 2 descriptions lost ....................... 41
5.4 Average Y-PSNR (in dB) for 3 descriptions lost ....................... 43
INTRODUCTION

Multiple Description Coding (MDC) is a type of video encoding which creates \( n \) outputs of coded video from a single raw video source. The different outputs, each one called a description, are individually packetized, encoded and sent to through the same, or separated, channel. The decodification process can be achieved with any subset of the descriptions to reconstruct the transmitted video. The distortion level of the reconstructed video depends on the number and quality of the descriptions available at the decoder. Anyway, when any of the descriptions are lost, the decoder can recover the transmitted video at lower quality than if it would have all the descriptions.

MDC particularly suits for real-time applications such as TV streaming over peer-to-peer (P2P) networks. The two main problems of video transmission in P2P networks are the different transmission bitrates among peers and the unstable nature of peer network connection. As MDC divides the information in \( n \) different channels, a single peer only needs to achieve a transmission rate that is a \( \frac{1}{n} \) factor lower than the original video encoded source. Furthermore, if a peer sending a description to another peer disconnects from the P2P network, the receiving peer still can decode (at lower quality) the video transmission as long as it receives data for at least one other peer. Unbalanced scenarios are common in xDSL access networks. The peer has different upload and download rate. The download link is, usually, greater than upload link.

This thesis studies different systems and scenarios in which MDC is applied. The system is implemented using two video codecs: MPEG-4 part 2 and H.264. The different peer transmission rates are simulated using two bitrate profiles. As it will be seen, the number of descriptions is set to 4. It also studies the recovered video quality when one, two or three descriptions are lost and different reconstruction methods to recover entire video stream.

This thesis is organized as follows. Chapter 1 provides an overview of the MDC system that has been built presenting the general scenario of the system, introducing the main software and hardware features. Chapter 2 deals with the video coding systems, MPEG-4 part 2 and H.264, which are compared in order to understand their performance. Chapter 3 explains different problems that must be treated at the receiver, explaining techniques of digital image processing like reconstruction and digital filters. Chapter 4 describes software system implementation. Chapter 5 analyzes the results of several testbed scenarios in which the system has been tested. Finally, chapter 6 presents the conclusions that result from this thesis.
CHAPTER 1. MDC SYSTEM ARCHITECTURE

1.1. Overview

Peer-to-peer (P2P) networks have played a key issue in the Internet growth. It represents a large portion of the total traffic in the global network. P2P networks are becoming useful networks to information distribution and sharing. If the traditional use of P2P networks has been file sharing, during the last years P2P has been used as a network for video transmission. The P2P topology is usually composed by a heterogeneous set of peers with different capabilities.

The correct transmission of the video stream over lossy environments has become an important issue and not only in the field of P2P networks. Multiple Description Coding (MDC) has emerged as a promising approach to enhance the error resilience of a video delivery system. MDC corresponds to the introduction of redundancy at the source level in order to generate multiple correlated streams with equal rate and equal importance that can be transmitted over different channels and decoded independently. So that each description alone provides low but acceptable quality and both descriptions together lead to higher quality [11] [2].

A primary reason for the increasing popularity of MDC is that it can provide enough quality without packet retransmissions. This implies MDC particularly attractive for real-time applications such as conference over portable devices or TV shows, for which retransmission is often not acceptable because it implies delays. In addition, it simplifies the network design. Do not need feedback channel or retransmissions and all packets can be treated equally [11].

As far as we know, the novelty of this work is the particular implementation of the entire MDC system described in [1] over Linux and in real time. The system takes any suitable device as input, such as a web-cam or a TV card, and applies MDC in real time. In this context, real time means that the time to process of each single video frame is lower than the generation time between two consecutive frames. The simulation of the network allows select between four different kind of received data patterns. With this four received patters, the receiver decodes each description and merge all descriptions to apply reconstruction methods.

The tests are done with two different bitrates: high and low profile. According to the conventional xDSL lines, the most common download rates are 2048 kbps and 1024 kbps. Each one of the four descriptions that the system generates has 512 kbps and 256 kbps, respectively. The combinations of different used codec, bitrate profile and description lost patterns will give the results of the tests.

In order to evaluate every combination and compare between them, the method used to give objective quality at the end of the process is Y-PSNR. Y-PSNR gives values between zero until 100 dB in order to know the level of similarity between two images. A higher value means high correlation (similarity) and a lower value means low correlation. Typical PSNR values range between 20 and 40 dB [10].
According to the MPEG committee recommendations, after each test done, the optimum reconstruction method is selected together with codec and bitrate profile to give the best performance in each case of description pattern lost.

1.2. General System Architecture

The general architecture of the system shown in figure 1.1 is formed two different parts: the capture and coding system and the receiver and decoding system, both connected through a communications network.

The capture and coding system is composed by the input of the system (the raw video source), at this case a web-cam, but it could be any other source such as a file or a TV computer card; FFmpeg software for manage the input and routing of the signal; a subsampler module which converts the input into four descriptions and FFmpeg coder for each description. Each description is sent to the receiver system.

The second part, the receiver and decoding system, is composed by FFmpeg as software to capture and decode the signal from each one of the received descriptions; an inverse subsampler module which reconstructs each one of the original frames from the $1 \leq n \leq 4$ descriptions received.

1.2.1. Capture and coding system

The capture and coding system is the first step in the acquisition system. This step allows to capture a sequence of frames from a web-cam, receive the entire sequence with FFmpeg software to adapt the colour space and frame rate to the Java source code in which the subsampler subsystem is coded, as it is shown in figure 1.2 and also the colour space used at each step.
Inside the Java source code, the system applies MDC and each output is passed as an input to a video encoder. Then, the encoded video is sent to the FFmpeg, which sends each description over to the network. The sketch of the system is showed in the figure 1.3.

![Figure 1.3: Subsampler block system](image1.png)

1.2.1.1. Input

The web-cam used is a Logitech QuickCam Express. The colour space at the output is Y’UV 4:2:2 planar. In this format each four bytes are two pixels. Each four bytes are two Y’s, a Cb and a Cr. Each Y goes to one of the pixels, and the Cb and Cr belong to both pixels. The Cr and Cb components have half the horizontal resolution of the Y component.

This colour space is used in PAL TV links and is included in ITU-601 standard. The figure 1.4 shows different kinds of chroma subsampling.

![Figure 1.4: Chroma subsampling patterns](image2.png)
1.2.1.2. **FFMPEG input management**

FFmpeg is a video and audio converter. It can also grab from a live audio/video source. In this step, FFmpeg converts video from live video source and send this converted video to FFserver. The conversion affects to the colour space. It converts from Y’UV 4:2:2 planar to YUV 4:2:0 planar. According to figure 1.4, the output video has six bytes for four pixels. Each six bytes are four to Y, one for U and the last one for V.

As the size of each frame is CIF (352 × 288), the frame rate is 25 frames per second, and the colour space is YUV 4:2:0 planar, the average bitrate at the output of FFserver is 30.4128Mbps. The complete equation is showed in the equation 1.1.

\[
\text{bitrate} = 25 \frac{\text{frames}}{\text{second}} \cdot 352 \times 288 \frac{\text{pixels}}{\text{frame}} \cdot \frac{6\text{bytes}}{4\text{pixels}} \cdot \frac{8\text{bits}}{\text{byte}}
\]

1.2.1.3. **FFserver streaming server**

FFserver can also stream audio and video data. It supports several live feeds, streaming from files and time shifting on live feeds (it is possible to seek to positions in the past on each live feed, provided you specify a big enough feed storage in a configuration file named as ffserver.conf). FFserver can runs in daemon mode. FFserver receives pre-recorded files or FFmpeg streams from some FFmpeg instance as input, then streams them over RTP/RTSP/HTTP.

The configuration of FFserver is stored in ffserver.conf. The parameters that we have configured are:

- **Port 8090**
- **BindAddress 0.0.0.0**
- **MaxClients 10**
- **MaxBandwidth 100000**
- **NoDaemon**

```
<Feed feed1.ffm>
    File /tmp/feed1.ffm
    FileMaxSize 100M
</Feed>
```

```
<Stream raw.yuv>
    Feed feed1.ffm
    Format rawvideo
    VideoSize 352x288
    Noaudio
</Stream>
```

The first part of ffserver.conf provides the general configuration of FFserver, setting that it listen for incoming connections at the port 8090 and all incoming IP connections are
allowed. We set the maximum number of simultaneous clients to ten and the maximum bandwidth for the FFserver to 100 Mbps. Finally, the process is not executed like as daemon. Then is configured the specific configuration of the server setting a buffer stream video, which its size is 100Mbytes. This option allows do time shifting with 100 Mbytes of buffer. Finally, we configure the output setting FFserver to output the video stream in raw video format. The raw format for FFserver is YUV 4:2:0 planar. The configured size is like a CIF.

1.2.1.4. Multiple Description Coding

MDC is implemented in Java. The source code connects with FFserver to get the video stream accessible to the Java code. The frames are processed one by one with the MDC algorithm described in [1]. The MDC decomposes the original image into subsets in the spatial domain. Each subset corresponds to a different description. The spatially or temporally adjacent video data samples are correlated.

The graphical idea is shown in the figure 1.5.

Each description has a quarter part of the entire luminance (Y) frame and the entire part of both chrominances (U and V). The resulting colour space in each description is YUV 4:4:4 planar, showed in the figure 1.4. The dimensions of luminance and chrominances are the same: a quarter part of luminance (Y) of the entire frame, \(176 \times 144\).

The output of each description is sent over different UDP connections, using stream video, to four different video encoders.
1.2.1.5. **Coder and transmission module**

The coder step has four identical processes, one to each description. FFmpeg reads the entire raw bit stream and it converts the input stream into coded output stream. The output stream is sent over the network. The used codecs are explained in the section 2.2..

1.2.2. **Receiver and decoding system**

The receiver and decoding system is the last process to obtain the recovered video stream. This step allows receiving video stream from another process in the same computer or from another computer over the network. The received video is decoded into the correct colour space and the inverse multiple description coding is applied. Then, the video can be played. These steps are showed in the figure 1.6.

![Figure 1.6: Reception block system](image)

1.2.2.1. **FFMPEG receiver and decode module**

Implemented in Java, four FFmpeg processes are running in the background of the system. FFmpeg is waiting for the reception of any video stream. The video stream is received from a fixed port. Each description is received with this method. Besides, FFmpeg decode each description into the correct colour space. The expected colour space for Java is YUV 4:4:4 planar. The dimensions of the image are QCIF (176 × 144). FFmpeg sends the video decoded to a specific port in local host over UDP.

1.2.2.2. **Inverse subsampler module over Java**

Inverse subsampler module does the inverse operation of the subsampler module. It implements the reconstruction of the entire video stream.

At the input, Java is listening in a specific UDP port. When some byte is received, Java knows that the video stream is in raw format, the colour space is YUV 4:4:4 planar and the size of the image is QCIF. Then the inverse subsampler is applied frame to frame. The graphical idea is showed in the figure 1.7.

The last step is the digital image processing. This step is explained in chapter 3.
Figure 1.7: Inverse Multiple Description Coding
CHAPTER 2. VIDEO CODING SYSTEM DESCRIPTION

This thesis compares the performance between different reconstruction mechanisms when Multiple Description Coding (MDC) is used. Besides, the comparative involves different codecs. The selected codecs are MPEG-4 part 2 and H.264. These are the most advanced video codecs at this moment. They have capabilities in order to maximize the compression with losses. Furthermore, the web-cam used in order to capture the video stream has the particularity that the output colour space format has gamma correction that are explained in this chapter for the web-cam and the two different used codecs.

2.1. Gamma correction

In analogue video devices the relation between the voltage applied to the Cathode Ray Tube (CRT) and the luminance (Y) created by the monitor is not linear. Their relation, according to [8] is expressed in the equation 2.1.

\[ B_{DISP} = k' \cdot (V_{DISP})^\gamma \]  

(2.1)

The ITU recommendation for black and white videos is \( \gamma = 2.2 \).

Each type of colour sub-pixel has gamma correction with different value compared with black and white value. These expressions are showed in equations 2.2, 2.3, and 2.4.

\[ R' = (R)^{1/\gamma_R} \]  

(2.2)

\[ G' = (G)^{1/\gamma_G} \]  

(2.3)

\[ B' = (B)^{1/\gamma_B} \]  

(2.4)

For colour television, the ITU recommendation is \( \gamma = 2.8 \).

The gamma correction is done in cameras because in the past, all TV receptors were CRT. The correction in cameras has the advantage that only needs one gamma correction for all the TV receptors.

The gamma correction effect is showed in the figure 2.1.

Due to the gamma correction, the TV shows in black and white in colour receivers will be dark, and vice versa. This is not a problem today.

The main problem today is the relation between voltage and luminance in any other kind
of screen; for example, LCD screens. This type of screen should apply another gamma correction in reception.

The gamma correction implies new luminance ($Y'$) signal. The new luminance is calculated by the corrected versions of RGB. These new expressions are shown in 2.5 for NTSC and PAL and 2.6 for HDTV.

$$Y' = 0.299 \cdot R' + 0.587 \cdot G' + 0.114 \cdot B'$$  \hspace{1cm} (2.5)

$$Y' = 0.2126 \cdot R' + 0.7152 \cdot G' + 0.0772 \cdot B'$$  \hspace{1cm} (2.6)

### 2.2. The MPEG-4 and H.264 Standards

The MPEG-4 part 2 (Visual) standard supports the coding and representation of visual objects with efficient compression and unparalleled flexibility. The diverse set of coding tools described in the standard are capable of supporting a wide range of applications such as efficient coding of video frames, video coding for unreliable transmission networks, object-based coding and manipulation, coding of synthetic and 'hybrid' synthetic/natural scenes and highly interactive visual applications. The figure 2.2 shows the object-based coding.

H.264 / MPEG-4 part 10 / AVC is, at this moment, the best video coder around the world. It is the natural evolution of MPEG-4 part 2 and H.263. Multi-frame prediction allows 32 frames between I frame and other P or B frame. Object-based coding is not supported. Quarter pixel motion estimation allow a more fine block matching estimation.

The table 2.1 [3] [7] summarises some of the main differences between the two standards.
This is not a complete comparison but it highlights some of the important differences between MPEG-4 part 2 and H.264.

According to ITU-T benchmark, the performance between H.264, MPEG-4 part 2, MPEG-2 and H.263 is showed in the figure 2.3.

As it can be seen in the figure 2.3 for the test sequence Foreman\(^1\), H.264 has the best performance. In this case, for Foreman in QCIF at 10 fps, at some bitrate, the Y-PSNR is better for H.264 codec. MPEG-4 part 2 and H.264 has similar performance. MPEG-2 is far away from the others codecs.

\(^1\)ITU-T provides some standard video sequences to compare performance between video coding systems. Foreman is one of these standard video sequences.
### Table 2.1: Summary of differences between MPEG-4 part 2 and H.264

<table>
<thead>
<tr>
<th>Comparison</th>
<th>MPEG-4 part 2</th>
<th>H.264</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported data types</td>
<td>Rectangular video frames and fields, arbitrary-shaped video objects, still texture and sprites, synthetic or synthetic-natural hybrid video objects, 2D and 3D mesh objects</td>
<td>Rectangular video frames and fields</td>
</tr>
<tr>
<td>Number of profiles</td>
<td>19</td>
<td>3</td>
</tr>
<tr>
<td>Compression efficiency</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Support for video streaming</td>
<td>Scalable coding</td>
<td>Switching slices</td>
</tr>
<tr>
<td>Motion compensation minimum block size</td>
<td>$8 \times 8$</td>
<td>$4 \times 4$</td>
</tr>
<tr>
<td>Motion vector accuracy</td>
<td>Half or quarter-pixel</td>
<td>Quarter pixel</td>
</tr>
<tr>
<td>Transform</td>
<td>$8 \times 8$ DCT</td>
<td>$4 \times 4$ DCT approximation</td>
</tr>
<tr>
<td>Built-in deblocking filter</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>License payments required for commercial implementation</td>
<td>Yes</td>
<td>Not (baseline profile); Yes (main and extended profiles)</td>
</tr>
</tbody>
</table>

Figure 2.3: Comparison between MPEG-2, H.264, MPEG-4 part 2 and H.264
CHAPTER 3. RECOVERING FROM DESCRIPTION LOST

All packets of each description contain data, which are necessary to reconstruct spatial regions, slides or macroblocks in the corresponding subframe. Due to unspecified problems in the network, some descriptions may be lost. For this reason is very important the recovering mechanisms. When the receiver has not all the descriptions, the system must implement an algorithm to decode the video sequence the best as it is possible.

Pixel error patterns may have different configurations, based on the number of descriptions lost. The five basic loss patterns are shown in figure 3.1.

Figure 3.1: Luminance loss pattern for a) one description lost, b) two descriptions lost, c) three descriptions lost

The proposed system is capable of recovering the lack of some description at the decoding moment. This is achieved with the help of the high spatial correlation that it exists between neighbouring pixels. The purpose of the algorithms presented in the following sections is the interpolation of missing pixels from the received ones. If Gaussian noise would have to be reconstructed, these algorithms would not be valid.

In this chapter, we are going to explain all algorithms used to reconstruct the lost descriptions, from the classical interpolation techniques, such as Near Neighbour Replication (NNR), to more complex non-linear interpolators. Classical linear interpolator tends to perform like as low pass filter to reconstruct lost pixels. This effect is visible around the natural edges. On the other hand, non-linear interpolators try to reproduce the missing pixels by
replicating natural edges. With this system, the effect is not of the low pass filter. The edges are more natural.

In the following sections, we detailed describe the proposed algorithms for MDC error concealment and compare their performance.

3.1. Digital image interpolation

Interpolation occurs when anyone tries to resize some image from one pixel grid to another. Image resizing is necessary to increase the total number of pixels, for example, to do zoom digital effect. Interpolation works by using known data to estimate unknown points. Image interpolation works in two directions. It tries to do a best approximation of a pixel from the surrounding values.

In this project, we use two kinds of interpolation: Near neighbour Replication (NNR) and Bilinear. The comparison between them are showed in the figure 3.2.

![Figure 3.2: NNR vs. Bilinear](image)

3.1.1. Near Neighbour Replication (NNR)

This method is the simplest method of image reconstruction. When some pixel is lost, the value of this pixel is the value of the nearest pixel. This scheme can be used from one description lost to three description lost. In the particular case of three descriptions lost, the restored image is the same than one description expanded.
3.1.2. **Bilinear**

Bilinear interpolator reconstructs the missing pixels by averaging between adjacent pixels. Depending on the loss pattern of 3.1 it can use from two pixels to four neighbouring pixels. For example, for one description lost, pixel $Y_0$ is restored by the averaging value between $Y_1$, $Y_3$, $Y_5$, and $Y_7$.

3.1.3. **Kernel implementation**

To do bilinear interpolation one alternative method is doing convolution. The convolution in two dimensions scenarios needs a kernel. The kernel is a grid in two dimensions. The kernel may be a matrix with even dimensions. Typically, the kernel size is $3 \times 3$. In this case, the kernel is showed in 3.1.

\[
\begin{bmatrix}
0.25 & 0.5 & 0.25 \\
0.5 & 1 & 0.5 \\
0.25 & 0.5 & 0.25
\end{bmatrix}
\]  

(3.1)

Regardless of, the convolution in the space domain is very expensive computationally. For an $n \times n$ kernel requires $n^2$ multiplications and additions per pixel. In the case of this thesis, the images has, approximately, $10^5$ pixels.

In this case, the optimum image processing is doing with Fourier Transform. In the transform domain, the filters change the module but maintain the phase. The product is between a set of real numbers and the real part of the Fourier Transform of the image.

3.2. **Edge Sensing**

Edge Sensing reconstruction algorithm (ES) is only valid when only one description is lost. This is due to in order to gradient computation involves too far from the current one and do not has better results than bilinear interpolator.

The algorithm 1 tries to reproduce the techniques of lossless image coding. The algorithm tries to look for the edges in horizontally and in vertically around of missing pixel. Recover the pixel taking the edge orientation into account.

The first condition detects an horizontal edge; the second one detects a vertical edge. If not, then the classical bilinear interpolator is used. The value of threshold $T$ is founded by experimentations in the [1]. For 8 bits/pixel images, the optimum value is about 50. With this value, the system achieves the maximum performance. The computational complexity is the same than bilinear interpolator.
Algorithm 1: Edge Sensing

$$\Delta H = |Y_1 - Y_5|;$$
$$\Delta V = |Y_3 - Y_7|;$$
if $$\Delta H < T$$ and $$\Delta V > T$$ then
$$Y_0 = (Y_1 + Y_5) / 2;$$
else if $$\Delta H > T$$ and $$\Delta V < T$$ then
$$Y_0 = (Y_3 + Y_7) / 2;$$
else
$$Y_0 = (Y_1 + Y_3 + Y_5 + Y_7) / 4;$$

3.3. Variable Number of Gradients

A set of eight gradients is computed from the 16 luminance pixel neighbourhood of $$Y_0$$. As it will be see, its performance is similar to that of Edge Sensing, at the expense of increased complexity. The proposed algorithm is extracted from [1]. The method to calculate the set of 8 gradients is showed in the equations 3.2, 3.3, 3.4, 3.5, 3.6, 3.7, 3.8, and 3.9.

$$G_1 = 2 \cdot |Y_1 - Y_3| + 0.5 \cdot (|Y_3 - Y_16| + |Y_2 - Y_5| + |Y_7 - Y_8| + |Y_7 - Y_15|)$$ (3.2)

$$G_2 = 2 \cdot |Y_2 - Y_6| + |Y_3 - Y_9| + |Y_1 - Y_{16}|$$ (3.3)

$$G_3 = 2 \cdot |Y_3 - Y_7| + 0.5 \cdot (|Y_1 - Y_2| + |Y_1 - Y_9| + |Y_4 - Y_5| + |Y_5 - Y_{10}|)$$ (3.4)

$$G_4 = 2 \cdot |Y_4 - Y_8| + |Y_3 - Y_{10}| + |Y_5 - Y_{11}|$$ (3.5)

$$G_5 = 2 \cdot |Y_1 - Y_5| + 0.5 \cdot (|Y_3 - Y_4| + |Y_3 - Y_{11}| + |Y_6 - Y_7| + |Y_7 - Y_{12}|)$$ (3.6)

$$G_6 = 2 \cdot |Y_2 - Y_6| + |Y_5 - Y_{12}| + |Y_7 - Y_{13}|$$ (3.7)

$$G_7 = 2 \cdot |Y_3 - Y_7| + 0.5 \cdot (|Y_1 - Y_8| + |Y_1 - Y_{14}| + |Y_5 - Y_6| + |Y_5 - Y_{13}|)$$ (3.8)

$$G_8 = 2 \cdot |Y_4 - Y_8| + |Y_1 - Y_{15}| + |Y_7 - Y_{14}|$$ (3.9)

Each gradient corresponds to a different direction ($$G_1 \iff \text{West}, G_2 \iff \text{North West}, \ldots$$).

A threshold value is computed like as equation 3.10, where $$\text{Min}$$ and $$\text{Max}$$ are the minimum and maximum values of the set of computed gradients.
\[
T = 1.5 \cdot \text{Min} + 0.5 \cdot (\text{Max} - \text{Min}) \tag{3.10}
\]

The subgroup \( I \) of gradients with absolute value less than \( T \) are selected. Every pixel is associated to every gradient, \( G_i \leftrightarrow Y_i \). The value of pixel \( Y_0 \) is computed by averaging value of neighbouring pixels which complies the last condition.

### 3.4. Post filtering

The main problem with the proposed MDC is a spatial granularity artifact in the reconstructed sequence. This problem is due to the four descriptions are coded independently. To reduce this effect, in this thesis we have implemented two kind of filters.

#### 3.4.1. 2D filter

This filter is extracted from the [1]. The intention of this filter is to reduce the effects of spatial granularity from flat regions in the reconstructed frames while preserving natural edges. It is very important to be able to distinguish between true edges and discontinuities due the granularity effect. To preserve the quality and non-distortion of the image, the true edges should not be filtered.

For each pixel, the algorithm computes where the edges are. The process to obtain edges in some image is the same process done in the Edge Sensing. They are defined two kind of pixels: the previous pixel \( x_{k-1} \) and the rear pixel \( x_{k+1} \). The evaluated pixel is \( x_k \). Note that the previous and rear pixel can be at the left and right of the pixel \( x_k \) or at the top or under the pixel \( x_k \).

A new parameter of threshold \( \beta \) is defined. \( \beta \) is determined by taking into account the quantization process of the video coding algorithm. The main idea is to perform filtering only if the difference between consecutive pixels is larger than a value implied by the coder quantization process. According to the [1], the value of \( \beta \) is found as shown in equation 3.11.

\[
\beta = 0.5 \cdot \left( 2^{QP/6} - 1 \right) \tag{3.11}
\]

The average values for QP are around 24 to 30.

The algorithm to decide if a pixel should be filtered is showed in the algorithm 2.

---

**Algorithm 2: 2D filter**

\[
\text{if } |x_k - x_{k-1}| / \beta \text{ and } |x_k - x_{k+1}| / \beta \text{ then } \]

\[
x_k = a \cdot x_{k-1} + \left( 1 - 2 \cdot a \right) \cdot x_k + a \cdot x_{k+1}
\]
3.4.2. Low-pass filter

Low-pass filter is a method to reduce the big differences between two pixels. These kinds of filters are used to remove high spatial frequency noise from a digital image. The noise is any kind of difference between original video stream and the recovered video stream. The low-pass filter averages out rapid changes in luminance (Y). This method calculates the average of a pixel and all of its neighbour pixels.

The filter can be done by a convolution kernel. The kernel used in this project is showed in 3.12.

\[
\begin{bmatrix}
\frac{1}{16} & \frac{2}{16} & \frac{1}{16} \\
\frac{2}{16} & \frac{4}{16} & \frac{2}{16} \\
\frac{1}{16} & \frac{2}{16} & \frac{1}{16}
\end{bmatrix}
\] (3.12)
CHAPTER 4. MDC SYSTEM IMPLEMENTATION

The programming language used in this thesis is Java. The main reason to choose Java is its multiplatform capabilities and it follows the object model paradigm. In this chapter we explain the different pieces of code created to implement the subsampling and error concealment in real time.

4.1. Capture and coder classes

4.1.1. YUVFrame

The figure 4.1 shows the YUVFrame class diagram. YUVFrame class is a representation of one frame in YUV colour space. The constructor allows to choose between 4:4:4 or 4:2:0 planar and set the dimensions of the frame (width and height). The dimensions of the frame, and its three different components can be obtained or set with the different getters and setters methods. The complete image is stored in a byte array. In addition, the class implements methods to convert the colour space between YUV 4:4:4 planar and YUV 4:2:0 planar.

<table>
<thead>
<tr>
<th>YUVFrame</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ YUVFrame(width : int, height : int, pix_fmt : String)</td>
</tr>
<tr>
<td>+ YUVFrame(width : int, height : int, pix_fmt : String, frame : byte[])</td>
</tr>
<tr>
<td>+ getWidth() : int</td>
</tr>
<tr>
<td>+ setWidth(width : int)</td>
</tr>
<tr>
<td>+ getHeight() : int</td>
</tr>
<tr>
<td>+ setHeight(height : int)</td>
</tr>
<tr>
<td>+ setPix_fmt(pix_fmt : String)</td>
</tr>
<tr>
<td>+ getPix_fmt() : String</td>
</tr>
<tr>
<td>+ getU() : byte[]</td>
</tr>
<tr>
<td>+ setU(u : byte[])</td>
</tr>
<tr>
<td>+ getV() : byte[]</td>
</tr>
<tr>
<td>+ setV(v : byte[])</td>
</tr>
<tr>
<td>+ setFrame() : byte[]</td>
</tr>
<tr>
<td>+ yuv444to420() : boolean</td>
</tr>
<tr>
<td>+ yuv420to444() : boolean</td>
</tr>
</tbody>
</table>

Figure 4.1: YUVFrame class diagram

4.1.2. YUVReader

YUVReader is an abstract class which has implemented the basic functions to read, write and send YUV image stream from different sources. This class avoids repeat code in several classes quite similar that can use these functions inheriting from the superclass YUVReader. The basic functions are protected and only inherited classes can use them.

The figure 4.2 shows all classes that inherit from YUVReader.
4.1.2.1. **YUVURLReader**

YUVURLReader class diagram is shown in the figure 4.3. This class has a main function that allows its execution as an application. The function `oneFrame()` allows to get a single YUV Frame from a URL.

4.1.2.2. **YUVFileReader**

This executable class allows to read a YUV video stream from a file. This file must be uncompressed and its colour space should be YUV 4:2:0 planar or 4:4:4 planar.

---

**Figure 4.4: YUVFileReader class diagram**

The class diagram is showed in the figure 4.4. The constructors need the frame dimensions of the video stream, the path and name of the file stored in the hard disk or other devices. The other constructor needs the same parameters and also the colour space.
The number of frames of the entire video stream can be obtained with the specific method `getNumFrames()`. Other auxiliary methods are implemented to close the video stream and obtain a new frame.

### 4.1.2.3. YUVRTPReader

YUVRTPReader reads the video stream from the specific local port that receive the specific program under Real Time Protocol (RTP).

![YUVRTPReader class diagram](image1)

<table>
<thead>
<tr>
<th>YUVRTPReader</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ YUVRTPReader(port : int, width : int, height : int)</td>
</tr>
<tr>
<td>+ oneFrame() : bytes[]</td>
</tr>
<tr>
<td>+ main(args : String[])</td>
</tr>
</tbody>
</table>

Figure 4.5: YUVRTPReader class diagram

As shows the figure 4.5, this class has the same methods than the YUVURLReader class. The methods have the same purpose, but have a different internal implementation to adapt the input stream to RTP.

### 4.1.2.4. YUVUDPReader

This class is very similar to YUVRTPReader. The difference is that it deals with UDP video streams. The figure 4.6 shows the class diagram of this class.

![YUVUDPReader class diagram](image2)

<table>
<thead>
<tr>
<th>YUVUDPReader</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ YUVUDPReader(port : int, width : int, height : int)</td>
</tr>
<tr>
<td>+ oneFrame() : byte[]</td>
</tr>
<tr>
<td>+ main(args : String[])</td>
</tr>
</tbody>
</table>

Figure 4.6: YUVUDPReader class diagram

### 4.1.3. Subsampler

Subsampler class is the core of the capture and coding system. This class implements the MDC subsampling. This class allows convert a YUV frame into four descriptions. The figure 4.7 shows the class diagram.

The constructor needs a YUVFrame class object to begin with the MDC process. The class loads the YUVFrame and subsample method runs the process. The class provide methods to obtain the four descriptions, the different components of the original YUVFrame, and to save each description into a specific path in the hard disk. This class is used by all main methods of the inhered classes from YUVRReader to subsample.
4.1.4. **Server**

Server class is showed in the figure 4.8. This class receive four descriptions from any of YUVReader family classes.

This is an executable class. The class implements the java core interface Runnable to be executed as a thread. Each description requires one instance of this class. This class needs different ports to listen and send video stream.

The incoming video stream arrives by UDP connection in YUV format. The class call to the encoder process to encode the video stream. Also, it saves the entire video stream in the hard disk and sends the output stream through an UDP connection. In order to send the video stream, the information needs be fragmented in multiple packets with the appropriate size, work done in a method of this class.

4.1.5. **Coder**

Coder class diagram is showed in the figure 4.9. Server class uses it for individual description codification with the appropriated codec.
This class only has one method. This method execute the specific FFmpeg line to encode each description video stream.

### 4.2. Receiver and decoding classes

#### 4.2.1. Receiver

This class is the core of the receiver and decoding system. The figure 4.10 shows the class diagram.

```
Receiver
+ Receiver(port1 : int, port2 : int, port3 : int, port4 : int, width : int, height : int)
+ oneFrame() : ArrayList<byte[]>
+ nextFrame() : ArrayList<byte[]>
+ save(a : ArrayList<byte[]>)
+ main(args : String[])
```

Figure 4.10: Receiver class diagram

Receiver class is an executable class which receives four descriptions from different UDP ports, decode and merge runs the inverse subsampler process in order to obtain the reconstructed video stream.

The constructor needs the dimensions of one frame and the four ports where each description will be received. The class provides different methods to manage the receiver process. Each group of four descriptions are grouped with specific structure to pass frame a frame and do correctly the decodification and merge processes. If the descriptions comes from hard disk, the class has a specific method to know if remain more frames to complete the entire process.

The class allows to save the restored video stream in the hard disk.

#### 4.2.2. Decoder

Decoder class is showed in the figure 4.11. The Receiver class in order to run the decoder process calls this class. The constructor needs the name of the file, and if the process comes from hard disk, the output port to where to send the decodec video stream and the specific colour space.

```
Decoder
+ Decoder(name : String, format : String, outputPort : int, type : String)
+ run()
```

Figure 4.11: Decoder class diagram
4.2.3. SubsamplerInv

SubsamplerInv receives four descriptions with YUV 4:2:0 planar or 4:4:4 planar colour space and merge it into a restored video stream. The complete class diagram is showed in the figure 4.12.

```java
public class SubsamplerInv {
    public SubsamplerInv(byte[] par, byte[] impar, byte[] imparPar, byte[] imparImpar, int width, int height) {
        // Constructor implementation
    }
    public YUVFrame getYUVFrame() {
        // Method to get YUVFrame
    }
    public byte[] getBytes() {
        // Method to get bytes
    }
    public BufferedImage convertToBufferedImage(int w, int h, byte[] data) {
        // Method to convert to BufferedImage
    }
    public byte[] convertToBytes(BufferedImage image) {
        // Method to convert to bytes
    }
}
```

Figure 4.12: SubsamplerInv class diagram

The constructor needs the four YUV byte arrays and the dimensions of the frame. The different methods permits get the YUVFrame or only the bytes.

Also, this class has implemented all digital image processing exposed in this thesis.
CHAPTER 5. EXPERIMENTAL RESULTS

Experimental results have been performed using ‘MSU Video Quality Measurement Tool’ in order to obtain Y-PSNR. The PSNR is an objective quality parameter which determines the quality between two images. The process to obtain two video streams to compare is done in two steps:

1. Encode original YUV 4:2:0 planar video stream with selected codec and decode it to YUV 4:2:0 planar. This sequence is showed in the figure 5.1.

2. Run the complete MDC process and when it is finished, each error concealment and post-filtering mechanisms are applied with different number of descriptions lost. The idea is showed in the figure 5.2 when one description is lost.

The lost of quality due to the MDC, to the reconstruction algorithm, and to the codec used, is measured by the Y-PSNR between the result of video stream showed in the figure 5.1 and result of each MDC reconstruction process with different number of descriptions lost showed in the 5.2.

To evaluate and take a reference with the comparison between different kind of methods and codecs, this master thesis takes the MPEG committee threshold. According to [9], the MPEG committee uses an informal threshold of PSNR = 0.5 dB to decide whether to incorporate a coding optimization because they believe that an improvement of that magnitude would be visible.
5.1. Encoding parameters

The software used to do the conversions between raw video and different kind of codecs is FFmpeg. Due the Internet speed line, the settings of the codecs are based in usual access networks bitrates. The most common Internet lines are asymmetric. The download bitrates are three or four times the upload bitrate. Two usual download bitrates are 1024kbps and 2048kbps. For this reason, the tests are done with the values of 256kbps and 512kbps for each description, respectively.

The distance, in number of frames, between two I frames (GOP) is 12 frames. The frame rate used is 25 frames per second. The codecs used are MPEG-4 part 2 and H.264. The wrapper formats to put the video stream are Audio Video Interleave (AVI) and Matroska Multimedia Container (MKV).

5.2. No Description lost

The reconstruction when no description is lost does not need any post filtering process. In this case, the receiver receives the entire image. The quality lost is only due to the encoder each description independently of the others descriptions. The expected results are, more bit rate, more Y-PSNR at the end of the process. The figure 5.3 shows the comparison between the two codecs and the two bitrates.

The figure 5.3 shows peak of Y-PSNR due the frames I. Other kinds of frames are between two consecutive peaks. In general terms, more bit rate implies more Y-PSNR. However, with the results shown in the figure 5.3 is very difficult say if x264 or MPEG-4 part 2 has better Y-PSNR.

The figure 5.4 shows the comparison, in terms of Y-PSNR, between the reconstructions with four descriptions at 256 kbps. The performance along all video stream is better for x264 codec except at the beginning of the sequence. At the beginning of the sequence,
the performance of the MPEG-4 part 2 is better. The possible explanation is that MPEG-4 part 2 has worst performance than x264. This situation does that the worse quality video, in average, in the first frames, when is compared with the reconstructed video stream (encoded independently) gives a better Y-PSNR during the 24 first frames. Note that the comparison is between encoded video sequence and received video stream.

The comparison between different codecs at 512 kbps has similar results than the 256 kbps performance. The difference is in terms of best Y-PSNR in both cases, but is not significant differences between video streams. The figure 5.5 shows this comparison.

The table 5.1 shows the comparison between the different codecs and bitrates in terms of Y-PSNR.

<table>
<thead>
<tr>
<th>MPEG-4 part 2</th>
<th>x264</th>
</tr>
</thead>
<tbody>
<tr>
<td>256 kbps</td>
<td>512 kbps</td>
</tr>
<tr>
<td>32.75027</td>
<td>35.38595</td>
</tr>
<tr>
<td>33.1767</td>
<td>33.1767</td>
</tr>
<tr>
<td>36.08608</td>
<td>36.08608</td>
</tr>
</tbody>
</table>

Table 5.1: Average Y-PSNR (in dB) for any description lost

According to average PSNR values, the best codec when all descriptions are received is x264. Between high and low bitrate, high bitrate gives more Y-PSNR than low bitrate. The difference between high and low bitrate profile is larger than 0.5 dB. For this reason and according to the MPEG committee threshold, the most appropriated bitrate is 512 kbps.

The frame number 205 is a good point to examine what happens with the images with high spatial frequency component. The difference at the frame 205 between MPEG-4 part 2 at 256 kbps per description and x264 at 512 kbps per description is showed in the figure 5.6.
Figure 5.4: Y-PSNR for any description lost at 256kbps

Figure 5.5: Y-PSNR for any description lost at 512kbps
The difference between them is very low for the human visual system. For this reason, the image is passed through the histogram equalization. Histogram equalization is a method to increase the contrast using the image histogram.

Figure 5.6: Original and difference between frame 205

The differences are very clear. The biggest difference is around the trees. The tree area has a high frequency component. The low profile has less bits to represent the same scenario than the high profile. The result is a poor definition, with a low profile, at the zones with high spatial frequency components.

5.3. One description lost

One description lost pattern is showed in the figure 3.1. In this case, are various types of robustness mechanisms to reconstruct the original video stream. The description of different mechanism of robustness is showed in the chapter 3.

This pattern allows uses Edge Sensing (ES) and Variable Number of Gradients (VNG). Both reconstruction mechanisms are designated only for one description lost. These mechanisms of robustness allows the 2D filter.

As showed in the figure 5.7, reconstructing the entire video stream with ES with or without 2D filter gives best Y-PSNR than other mechanism shown. The reason of these results are that the others mechanisms are simpler than others like Bilinear interpolator. ES and VNG takes into account the orientation of the edges to minimize the effect due to the independent codification.

The number of techniques compared in the figure 5.7 is high. For this reason, the best reconstruction mechanism with best performance is not clear. The figure 5.8 shows the comparison between ES and VNG. The result is that ES gives better Y-PSNR than VNG in all video stream.

Figure 5.9 shows the comparison between ES without post filtering and ES with 2D filter. The performance is better when the post filtering is applied. The difference between them
Figure 5.7: Y-PSNR MPEG-4 part 2 - One description lost at 256kbps/description

Figure 5.8: Y-PSNR MPEG-4 part 2 - One description lost at 256kbps/description. ES and VNG
Experimental Results

is lower than in the comparison shown in figure 5.8.

Figure 5.9: Y-PSNR MPEG-4 part 2 - One description lost at 256kbps/description. ES with and without post filtering

The comparison between the two reconstruction mechanisms with post filtering is shown in the figure 5.10. ES with post filtering gives more Y-PSNR than VNG with post filtering. The difference between both methods is showed in the figure 5.11. The difference is always positive. This means that ES with post filtering gives better Y-PSNR along all video stream. The differences goes from around 0 up to 3 dB, approximately. At the frame 181, the difference between ES and VNG is about 2.8 dB. The average value of difference between them is 0.8 dB. According to the MPEG committee, this magnitude would be visible because is larger than 0.5 dB.

The comparison between exposed reconstruction mechanism with high bitrate and MPEG-4 part 2 and low and high bitrate with x264 codec has the same results than with MPEG-4 part 2 at 256 kbps per description. ES with 2D filter gives the best performance versus Bilinear interpolation and VNG with or without filter. The next figures of performance will be to study which are the differences between different bitrates and codecs.

The figure 5.12 shows the performance between the four profiles we have used. x264 with 512 kbps per description gives the best Y-PSNR.

At the second frame, MPEG-4 part 2 with low bitrate gives the best performance. x264 with low bitrate gives the worse performance at the second frame. The comparison between both is shown in the figure 5.13. MPEG-4 part 2 gives better Y-PSNR values at the first part of the video stream and when the sequence changes between human face to another scenario, x264 gives best quality. MPEG-4 at low bitrate gives best performance for static scenes and x264 gives best performance for scenes with high temporal movement. Ac-
Figure 5.10: Y-PSNR MPEG-4 part 2 - One description lost at 256kbps/description. Post filtering

Figure 5.11: Y-PSNR MPEG-4 part 2 - One description lost at 256kbps/description. Difference between ES and VNG with post filtering
According to 2.2., the motion vector accuracy is better for x264 than MPEG-4 part 2. For this reason, x264 gives best performance in scenarios with movement. Otherwise, for static scenarios with natural objects, like a face, the MPEG-4 part 2 gives better results.

The figure 5.14 shows the differences between the frames number two from MPEG-4 part 2 at 256 kbps per description and x264 at 256 kbps per description. As shows the figure, the main number of differences is in the natural edges like the face edges and scenario forms.

Like the figure 5.6, the figure 5.14 is passed through the histogram equalization in order to increase the contrast.

As shows the figure 5.15, in general, the performance is better for x264 at 512 kbps in all video stream. The performance increases when the video sequence changes from human face to other scenario.

The complete values of average Y-PSNR are showed in the table 5.2. This table shows the values of different reconstruction mechanisms, post filtering, bitrate and codec used.

According to the average values of Y-PSNR, the reconstruction mechanism with best performance is ES and 2D filter. x264 gives the best performance in all reconstructions mechanisms when one description is lost. The difference between high and low bitrate is higher than 0.5 dB [9]. For this reason, the most appropriated bitrate is 512 kbps.
Figure 5.13: Y-PSNR for one description lost - ES with post filtering at 256kbps/description

Figure 5.14: Original and difference between frame 2
Figure 5.15: Y-PSNR for one description lost - ES with post filtering at 512kbps/description

<table>
<thead>
<tr>
<th>Reconstruction mechanism</th>
<th>MPEG-4 part 2</th>
<th>x264</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bilinear interpolation</td>
<td>31.7037</td>
<td>33.1085</td>
</tr>
<tr>
<td>Edge Sensing</td>
<td>32.0594</td>
<td>33.69085</td>
</tr>
<tr>
<td>Variable Number of Gradients</td>
<td>31.25846</td>
<td>33.10887</td>
</tr>
<tr>
<td>Edge Sensing and 2D filter</td>
<td>32.10025</td>
<td>33.77494</td>
</tr>
<tr>
<td>Variable Number of Gradients and 2D filter</td>
<td>31.28478</td>
<td>33.17801</td>
</tr>
</tbody>
</table>

Table 5.2: Average Y-PSNR (in dB) for 1 description lost

5.4. Two descriptions lost

This section shows the comparison between reconstruction mechanisms with two descriptions at the receiver. The figure 3.1 shows this pattern.

The mechanisms of robustness used to do this section are NNR and Bilinear interpolation. Post filtering filters are not used in this section because gives worst performance than without it.

The figure 5.16 shows the comparison between NNR and Bilinear interpolation when the used codec is MPEG-4 part 2 and the bitrate of each description is 256 kbps.

As the figure 5.16 shows, Bilinear interpolation gives more Y-PSNR than NNR in all video stream for MPEG-4 part 2 codec and each description at 256 kbps.
The figure 5.17 shows the comparison between different reconstruction mechanisms at 512 kbps and MPEG-4 part 2. The performance between NNR and Bilinear interpolation are very similar to the case with low bitrate, but the global levels of Y-PSNR are higher.

The figures 5.18 and 5.19 shows the performance between NNR and Bilinear interpolation with x264 codec.

The results are similar to the MPEG-4 part 2 codec used. Bilinear interpolation gives best Y-PSNR values than NNR. More bitrate gives more Y-PSNR values.

In all combinations of codecs and bitrates, Bilinear interpolation gives better performance than NNR. For this reason, the figure 5.20 shows the comparison between MPEG-4 part 2 and x264 at low and high bitrates profiles.

The performances between low and high bitrates are very similar. High bitrates gives best Y-PSNR in all video stream.

The difference between MPEG-4 part 2 and x264 is not clearly. The figure 5.21 shows the difference between x264 and MPEG-4 part 2 at 512 kbps each description. Positive values means best x264 performance and negative values means best MPEG-4 part 2 performance.

At the first frames, MPEG-4 part 2 gives best Y-PSNR values. The reason is that MPEG-4 part 2 has support for natural video objects and at the first frames, the main object in the video stream is a human face. After this, the human face change to another scenario and x264 has better support with best motion approximation. The average value of the difference is 0.09 dB.
Figure 5.17: Y-PSNR MPEG-4 part 2 - Two descriptions lost at 512kbps/description

Figure 5.18: Y-PSNR x264 - Two descriptions lost at 256kbps/description
Figure 5.19: Y-PSNR x264 - Two descriptions lost at 512kbps/description

Figure 5.20: Y-PSNR for two descriptions lost - Bilinear interpolation
Figure 5.21: Y-PSNR for two descriptions lost at 512kbps/description - Bilinear interpolation. Difference between x264 and MPEG-4 part 2

The figure 5.22 shows the difference between x264 and MPEG-4 part 2 at 256 kbps per description. Positive values means implies best x264 performance and negative values implies best MPEG-4 part 2 performance.

At the first part of the video stream, MPEG-4 part 2 gives best Y-PSNR performance because it gives support to natural video objects. After the first frames, x264 gives better support. The average difference between them is about 0.073 dB.

The table 5.3 shows the comparison between bitrates, codecs and reconstruction mechanisms for 2 descriptions are lost in terms of average Y-PSNR.

<table>
<thead>
<tr>
<th>Reconstruction mechanism</th>
<th>MPEG-4 part 2</th>
<th>x264</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>256 kbps</td>
<td>512 kbps</td>
</tr>
<tr>
<td>NNR</td>
<td>28.98405</td>
<td>29.47317</td>
</tr>
<tr>
<td>Bilinear interpolation</td>
<td>30.84179</td>
<td>31.90869</td>
</tr>
</tbody>
</table>

Table 5.3: Average Y-PSNR (in dB) for 2 descriptions lost

According to the average values of Y-PSNR, the reconstruction mechanism to 2 descriptions lost with best performance is Bilinear interpolation. x264 gives the best performance for Bilinear interpolation and MPEG-4 part 2 gives best performance when NNR reconstruction mechanism is used. The differences between high and low bitrate with Bilinear interpolation reconstruction mechanism and x264 codec used is larger than 0.5 dB [9]. For this reason, the most appropriated bitrate is 512 kbps.
Three descriptions lost pattern is the worst case. The figure 5.23 shows the comparison between the three reconstruction mechanism when the used codec is MPEG-4 part 2 and the bitrate per description is 256 kbps. In general terms, reconstruction the entire video stream with bilinear interpolator gives best Y-PSNR than both mechanisms with NNR. The reason of these results is that the NNR is the worst and easiest reconstruction method. NNR only copies the more nearest pixel. Bilinear interpolator mechanism takes into account the value of surrounding pixels. The effect is the best Y-PSNR because, in space domain, the neighbour pixels has high level of correlation.

Apply the low-pass filter to the video stream restored by NNR mechanism gives good results increasing the average Y-PSNR. Low-pass filter introduces the advantage that needs the value of surrounding pixels. The high correlation between adjacent pixels gives better Y-PSNR than NNR case.

The segment between frames 185 and 221 is showed in the figure 5.24. In these frames, the performance between the three mechanisms of robustness is inverted. The reason is the sequence in the video stream. The sequence of images in these frames is very changing because the camera changes the plane between a human face to other scenario. When that happens, the macroblocks in compressed video stream move very fast from right to left. The correlation between adjacent pixels, in space and time domain is lower than when is a static image.

Figure 5.22: Y-PSNR for two descriptions lost at 256kbps/description - Bilinear interpolation. Difference between x264 and MPEG-4 part 2
The performance between the three reconstruction mechanisms when the bitrate is 512 kbps and the used codec is MPEG-4 part 2 is showed in the figure 5.25. Three mechanisms give better Y-PSNR than in the case of lower bitrate per description. All sequence gives the same performance between different reconstruction mechanisms than in the case of 256 kbps.

The figure 5.26 shows the comparison at the case when the used codec is x264. The tendency between different reconstruction mechanisms is like the MPEG-4 part 2 used codec.

The version of high bitrate with x264 used codec is showed in the figure 5.27. The performance in three reconstruction mechanisms is better than the case of lower bitrate.

With these figures, the comparison between different codecs is not easy. The table 5.4 shows the comparison between the different reconstruction mechanisms and codec used in terms of the average bitrate.

<table>
<thead>
<tr>
<th>Reconstruction mechanism</th>
<th>MPEG-4 part 2</th>
<th></th>
<th>x264</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>256 kbps</td>
<td>512 kbps</td>
<td>256 kbps</td>
<td>512 kbps</td>
</tr>
<tr>
<td>NNR</td>
<td>26.02772</td>
<td>26.02814</td>
<td>25.84622</td>
<td>25.90085</td>
</tr>
<tr>
<td>Bilinear interpolation</td>
<td>27.06114</td>
<td>27.28737</td>
<td>27.02631</td>
<td>27.24781</td>
</tr>
</tbody>
</table>

Table 5.4: Average Y-PSNR (in dB) for 3 descriptions lost

According to average PSNR values, the best reconstruction mechanism when only arrive one description at the receiver is 'Bilinear interpolation'. MPEG-4 part 2 gives the best
Figure 5.24: Foreman video stream. High moving sequence
Figure 5.25: Y-PSNR MPEG-4 part 2 - Three descriptions lost at 512kbps/description

Figure 5.26: Y-PSNR x264 - Three descriptions lost at 256kbps/description
performance, in all reconstruction mechanisms when three descriptions are lost. But the difference between high bitrate and low bitrate is lower than 0.5 dB. For this reason and according to the MPEG committee threshold, the most appropriated bitrate is 256 kbps.
CHAPTER 6. CONCLUSIONS

In this work, we have studied a practical implementation of a MDC system in real time and its different performance between different implementation methods. The MDC in real time is working with good mechanisms of robustness.

The different reconstruction methods that have been implemented and their performances have been analyzed. The codec with best performance in all cases is H.264. MPEG-4 part 2 only gives better performance than H.264 when only arrives one description at the receiver. According to the level of development in both codecs, the obtained results are as they were expected.

The bitrate that gives best performance in three of the four studied cases is 512 kbps. It is obvious that more bitrate gives more quality, but it is not always true. The particular description lost pattern when three descriptions are lost is a exception.

The mechanism of robustness with best performance is one of the most complicated algorithms. ES gives the best performance when one description is lost. The combination with 2D filter at the post filtering block gives more quality to the reconstructed video stream. The method with low complexity like NNR gives a poor efficiency. Adding post filtering block to the NNR gives, in some cases, better performance in terms of Y-PSNR. The Bilinear interpolation algorithm, gives a good performance in all tests. This mechanism gives the best performance in two of four cases.

6.1. Environmental impact

Since it is a software development, the only environmental impact lies in the energetic expenses generated to carry out this project.


TITLE : A Real-Time Multiple Description System

MASTER DEGREE: Master of Science in Telecommunication Engineering & Management - MASTEAM

AUTOR: Manuel Bermejo Plana

DIRECTOR: Sergio Machado Sánchez

DATE: January 14, 2010
APPENDIX A. REAL-TIME CAPTURE AND DECODER

A.1. Objective

The objective is capture and decodes the video stream from an input device, web-cam for example, in real time.

A.2. FFmpeg

According to [6], FFmpeg is a complete, cross-platform solution to record, convert and stream audio and video. It includes libavcodec - the leading audio/video codec library.

The project is made of several components:

- ffmpeg is a command line tool to convert multimedia files between formats.
- ffserver is a multimedia streaming server for live broadcasts.
- ffplay is a simple media player based on SDL and the FFmpeg libraries.
- libavutil is a library containing functions for simplifying programming, including random number generators, data structures, mathematics routines and much more.
- libavcodec is a library containing decoders and encoders for audio/video codecs.
- libavformat is a library containing demuxers and muxers for multimedia container formats.
- libavdevice is a library containing input and output devices for grabbing from and rendering to many common multimedia input/output software frameworks, including Video4Linux, Video4Linux2, VfW, and ALSA.
- libswscale is a library performing highly optimized image scaling and colour space/pixel format conversion operations.

A.3. System’s configuration

This section explains the main configuration used to do the capture and decoder in real time.
A.3.1. Basic configuration

A.3.1.1. System configuration

The specifications of the computer:

- CPU: Intel (R) Core(TM) 2 Duo CPU E6750 @ 2.66GHz.
- Motherboard: ConRoe1333-D667 with Intel Chipset.
- Memory: Kingston DDR2 RAM with 2048 MBytes.
- Graphics: Intel(R) 82945G Express Chipset Family with 224 MBytes of memory.
- Network device: Realtek RTL8139/810x Family Fast Ethernet NIC.

A.3.1.2. Operating System

The operating system used to do this is Debian 5.0 [5]. Debian is developed by the Debian Project. The Debian Project is an association of individuals who have made common cause to create a free operating system. This operating system that we have created is called Debian GNU/Linux, or simply Debian for short.

Debian systems currently use the Linux kernel. Linux is a piece of software started by Linus Torvalds and supported by thousands of programmers worldwide.

A.3.1.3. Input system’s

The input system’s is a web-cam. The web-cam is Logitech Inc QuickCam Express and its ID is 046d:0840.

A.3.1.4. Debian Multimedia

In addition, Debian Multimedia packages may be installed. According to [4], the Debian Multimedia project aims to make Debian a good platform for audio and multimedia work. This basically means packaging/maintaining multimedia applications and libraries, and collaborating with other maintainers or teams in order to improve audio/video support in Debian.

The Debian Multimedia team is currently merging with the Debian Multimedia Packages team. Both teams are converging into one, the Debian Multimedia Maintainers team. This means that there is some duplicate infrastructure that should converge on a best-effort basis.
A.3.2. FFServer configuration

FFServer needs some specification at its configuration file. The file name is ‘ffserver.conf’. The entire configuration is showed in the following lines.

Port 8090 # Port on which the server is listening
BindAddress 0.0.0.0 # Address on which the server is bound
MaxClients 10
MaxBandwidth 100000
CustomLog - # Access log file
NoDaemon # Suppress that if you want to launch ffserver as a daemon

# Definition of the live feeds. Each live feed contains one video
# and/or audio sequence coming from an ffmpeg encoder or another
# ffserver. This sequence may be encoded simultaneously with several
# codecs at several resolutions.
<Feed feed1.ffm>
File /tmp/feed1.ffm
FileMaxSize 50M
</Feed>

<Stream raw.yuv>
Feed feed1.ffm
Format rawvideo
VideoSize 352x288
Noaudio
</Stream>

A.3.3. Commands

In different shell instances, in order to run the capture and decoder system, launch the FFServer and FFmpeg processes.

$ ffserver
$ ffmpeg -r 25 -s 352x288 -f video4linux -i /dev/video0
    http://localhost:8090/feed1.ffm

With these command lines, the video stream is up. In order to obtain the video stream, follow the following line with any web browser software:

ipVideoServer:portVideoServer/raw.yuv

For example, for this example, the complete URL is:

192.168.1.2:8090/raw.yuv
A.3.3.1. Options

The specifications of all options of ran lines:

- `-r 25`: 25 frames per second.
- `-s 352x288`: set frame size. This frame size is equivalent to ’cif’.
- `-f video4linux`: the input feed to FFmpeg. Video4Linux is the multimedia interface for Linux.
- `-i /dev/video0`: input filename. It corresponds to web-cam input device.

A.3.3.2. Transcodificate video with ffmpeg

Maybe, YUV video file cannot be reproduced with some media players. In order to see the video stream, FFmpeg can transcodificate the entire video stream into MPEG-4 part 2 with AVI file extension. The command used is showed at the following line.

```
$ ffmpeg -s 352x288 -i inputFile.yuv outputFile.avi
```