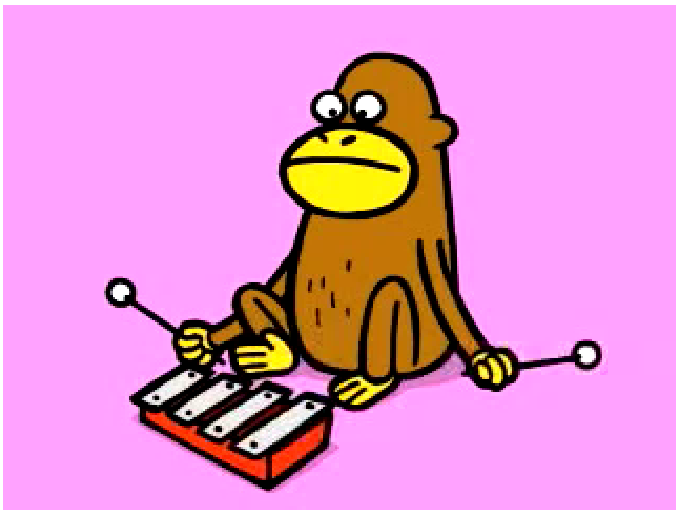
Compression, analysis and visualization of multimedia contents

Labs 5,6 and 7 report

Francisco Santos García — EIT Data Science
Francisco de Borja González González — EIT Data Science

January 8, 2019



POLYTECH NICE-SOPHIA

1 Introduction

This document describes the most relevant information of the practical work "Motion estimation and compensation-video compression", which consist on three labs where we go through the different stages that make up the video coding workflow.

Motion estimation is a process which is used in video compression in order to reduce temporal redundancy between adjacent frames. It examines the movement of objects in an image sequence to try to obtain motion (velocity) vectors representing at best the estimated motion

The main objective of this document is to present the results obtained for each lab. We will apply what we have learned in the theoretical classes in a real practical case. Each lab also has their own specific goals.

In order to achieved those objectives, we will follow the Video Coding Workflow. The figure below describes the General Workflow for a video coding.

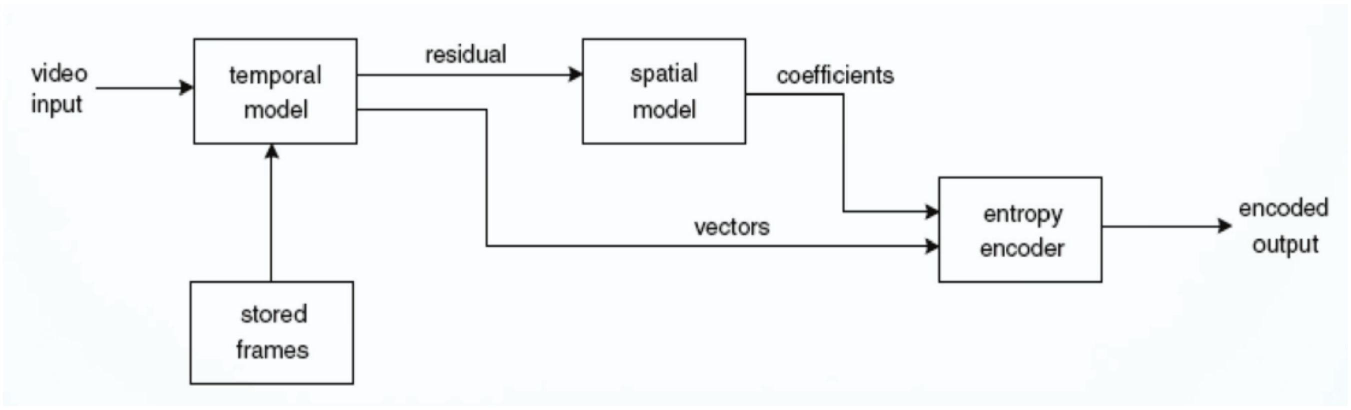


Figure 1: Video Coding - General Workflow

The two main components of the figure above are the temporal model and the spatial model. In the first one, we calculate the motion estimation and the motion compensation of two consecutive frames of the video to obtain the motion vectors. While in the second one, we apply a transformation of a frame, for example the DCT (Discrete Cosine Transform) to obtain the coefficients. Finally, in the last stage of the workflow, we calculate the entropy of the coefficients and the motion vectors and we obtain the encoded video.

For all the labs of this assignment, we have used Python as programming language and the well known library OpenCV to deal with the input video and the different frames (images). In this document, we will include some code snippets of the most representative functions.

2 Lab5

During this lab we will perform a motion estimation by implementing a block-matching algorithm of two consecutive frames of a video. We will call those frames F_R (Reference Frame) and F_C (Current Frame), being F_R=F_C-1.

As a result of applying our block-matching algorithm, we will obtain the motion vectors between the two frames. The motion vector, is the key element in the motion estimation process and it is used to represent a macro-block on a frame based on the position of this macro-block in other frame, called the Reference Frame. The figure below illustrates this process.

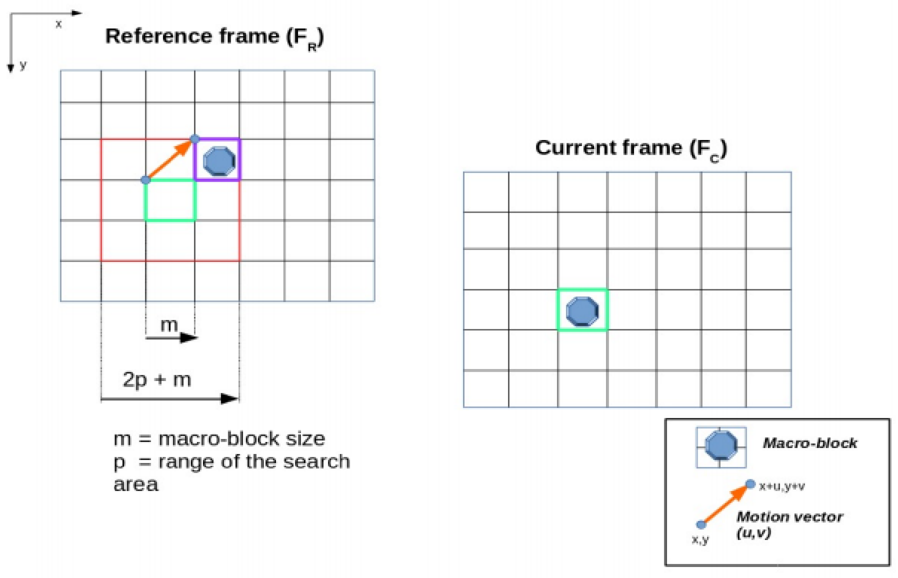


Figure 2: Block-Matching and motion vectors

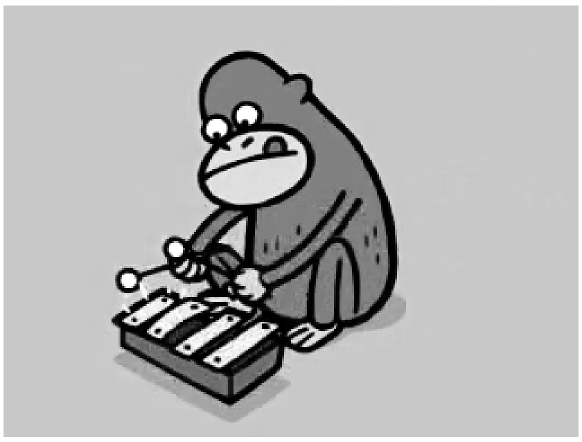
For this purpose, we have selected the proposed video, called: *xylophone.mp4*. Some of the properties of this video are:

- Duration: 10 seconds
- Dimensions (Resolution): 320x240
- Video Codec: H.264
- Framerate: 15 frames per second
- Bitrate: 263 kbps
- Number of frames: 150 frames
- Original video size: 430.6 KB

The first thing we have to do to build our block-matching algorithm is to import the video and select two consecutive frames. For this Lab we have selected the frames 48 (F_R) and 49 (F_C), as we have appreciate a clear difference between both frames after watch the video several times.

Listing 1: Reference Frame and Current Frame selection

```
#import requiered libraries
import cv2

#Capture the video with OpenCV
cap = cv2.VideoCapture("./xylophonemp4.mp4")

#Set the Current Frame and the Reference Frame
current_frame=49
reference_frame=current_frame-1

#Set and read the Reference Frame from the input video
cap.set(1,reference_frame);
retRF, referenceFrame = cap.read()

#Convert the Reference frame to grayscale image
gray_RF = cv2.cvtColor(referenceFrame, cv2.COLOR_BGR2GRAY)

#Set and read the Current Frame from the input video
cap.set(1,current_frame);
retCR, currentFrame = cap.read()

#Convert the Current frame to grayscale image
gray_CF = cv2.cvtColor(currentFrame, cv2.COLOR_BGR2GRAY)

#Save the Reference Frame and the Current Frame as png images
cv2.imwrite("./referenceFrame.png", gray_RF)
cv2.imwrite("./currentFrame.png", gray_CF)
```


(a) Frame 48. Reference Frame (F_R)

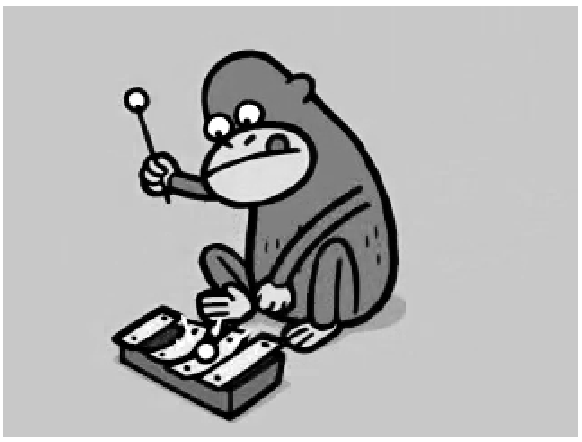

(b) Frame 49. Current Frame (F_C)

Figure 3: Reference Frame and Current Frame selection

Following the *Figure 2*, we have to define a macro-block size (m) in pixels and a search parameter (p). With this two variables we can define the area of pixels around around the position defined by the current macro-bloc, within which we search for the best mach.

The best match evaluates the minimum cost function between the corresponding block at each possible location, this means that for every macro-block of F_C, we will search for the best match in F_R, in the range of the search area. For this, we have used the full search method.

We have used the proposed metric "Mean Square Error" (MSE) to evaluate the cost function. In python, we have computed the MSE between two frames (images) as:

Listing 2: MSE between two frames (images) in Python

```python
def mse_frames(frame1, frame2):
    return ((frame1 - frame2)**2).mean(axis=None)
```

For each macro-block, the areas with the minimum MSE are the ones that we take to represent the motion estimation between the Reference Frame and the Current Frame. Therefore, the motion vectors is composed with the minimum cost function.

The results obtained for a macro-block size, m = 16 (blocks of size mxm pixels) and a search parameter p = 4, for the consecutive frames 48 and 49 are shown in the following quiver plot:

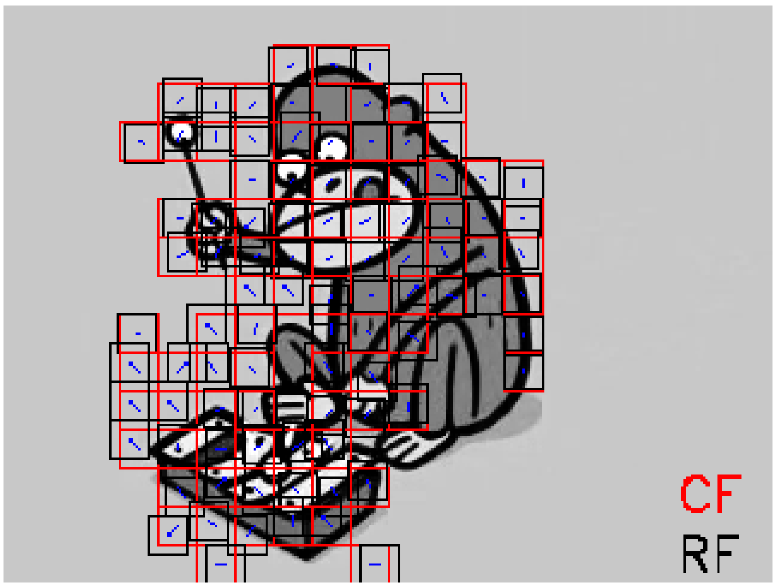


Figure 4: Quiver plot, motion estimation between frames 48 and 49

If we compare the two images shown in the *Figure 3* and the results obtained above, we can perceive how the monkey has leaned slightly downwards. Also, we can perceive a huge movement on the two instruments that the monkey is carrying on his hands. The motion vectors are shown as arrows that follow the direction of the movement.

If we increase the value of p, (for example p=32), we will get bigger blocks and the precision of our motion estimation will be worse, although our algorithm will be faster. While if we decrease the value of p (for example p=4), we will obtain smaller blocks, the precision of our algorithm will be better, but slower because we are estimating the movement of many more blocks than in previous occasions.

3 Lab6

After performing motion estimation with a block-matching algorithm for two consecutive frames of a video F_R and F_C we will build up a new frame called F_{CC} Motion compensated Frame.

3.1 Creation of Motion Compensated Frame F_{CC}

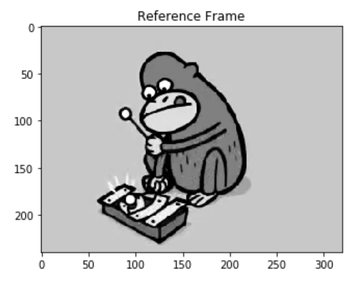
From the lab 5 we have defined the search parameter p for the surrounding pixels we want to scan. We have also defined the size of blocks we want to make out of the images, m. After obtaining the best match from every macro-block of F_C with F_R we will use the motion vectors to build up a new frame. To do this, we will iterate block by block by each row of the image and we will store the movement of each block on a list. This will form a matrix with the movements of each block.

After that, for creating the motion compensated frame F_{CC} we will iterate block by block the reference frame F_R and we will place them in the place that the motion vectors tell us:

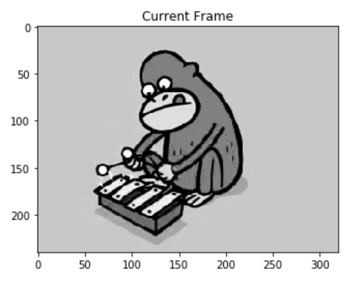
Listing 3: Building up of the motion compensated frame

```
for height in range(blocks_height):
    for width in range(blocks_width):
        print(”We␣are␣in␣y=”,h ,”x=”, w)
        y0, y1 = height*m, y0+m
        x0, x1 = width*m, x0+m
        movement_block = motion_vectors[height][width]
        print(”Block␣is␣moving”, motion_vectors[height][width])
        yshift ,xshift = movement_block[1], movement_block[0]
        frame_motion_comp[y0:y1, x0:x1] = frame_ref_gray[y0-yshift:y1-yshift , x0-xshift:
```

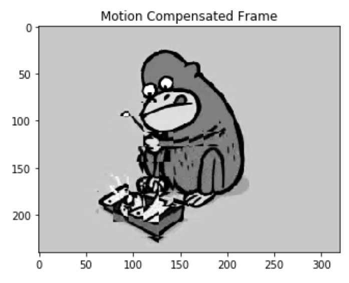
As we can see in the figures below, it is not exactly the same as the current frame. Since we have chosen two frames with big movement between them it is harder to make a good motion compensated frame.



(a) Reference (F_R)



(b) Current (F_C)

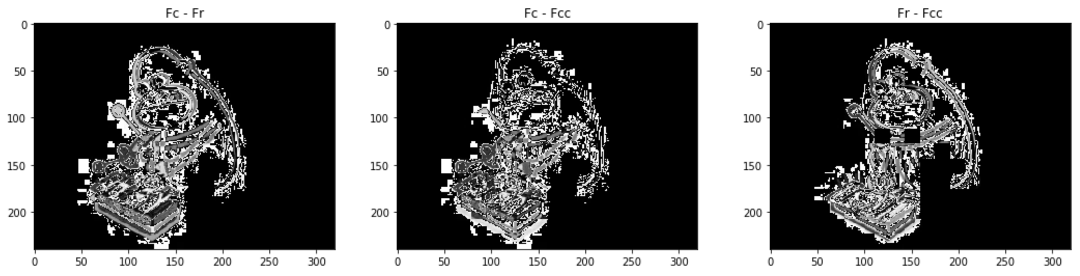


(c) Compensated (F_{CC})

Figure 5: The building of the Motion Compensated Frame

3.2 Differences between frames

What we can do now after we have our 3 study frames is to compare the differences of values between each of them. This is a task that might be useful not only on compression tasks but also it has many applications on image processing tasks. The difference between F_C and F_{CC} is also called E_{res}.



(a) F_C - F_R (b) F_C - F_{CC} (c) F_R - F_{CC}

Figure 6: Differences between Frames

3.3 Average motion compensated error

This section is just about coding the motion compensated error using the mean absolute error expression. We have computed it as follows:

Listing 4: Building up of the motion compensated frame

```
def mae(f1 , f2):
    df = pd.DataFrame({'f1':np.array(f1.flatten(), dtype = 'f'),
'f2':np.array(f2.flatten(), dtype = 'f')})
    df_error['error'] = (df_error['f1'] - df_error['f2']).abs()
    return sum(df_error['error']) / len(df_error)
```

If we calculate the error by pairs:

MAE for F_C and F_R	MAE for F_C and F_{CC}	MAE for F_R and F_{CC}
16.82	9.80	13.40

Table 1: MAE between pairs of frames.

We can observe that the difference between the current and the reference is higher than than the current and the predicted. This also is going to depend on the chosen frames, since we are taking 2 with big differences (the monkey moves the most), it is going to be more differences between frames and the prediction is going to be more difficult.

3.4 Perform motion estimation and motion compensated prediction on different pairs of consecutive frames and comment your observations.

For this last part, we need to store the motion vectors to compare and study the motion estimation vs the prediction on several frames. Since it is a long process, we have selected the first 20 frames and different rates of pixel searching p to make a deeper study of the results. The motion estimation is how much a frame moves respect its previous.

Firstly, we are going to compare the different processing times for different search parameters:

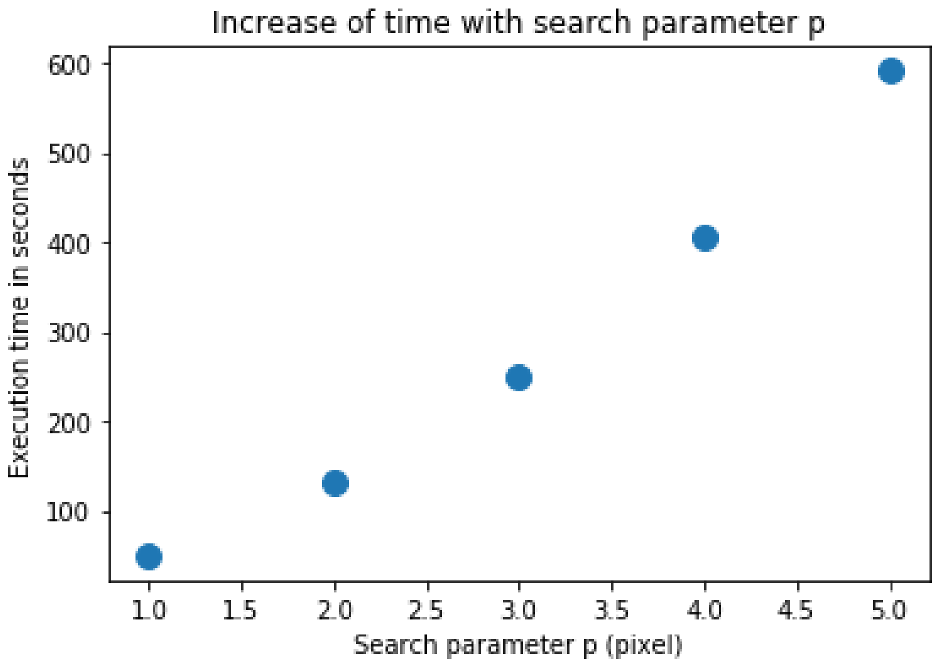


Figure 7: Execution Time for different values of p.

We can infer from the upper plot that increasing the pixel search is very costly in terms of time (following an exponential law) so, when you are working on more frames (in this case we are using 20) one must think about if it is worth it to do it in terms of time - efficiency.

In this section, we have also computed the motion compensated error for each frame considering as explained before several values of:

frame.png frame.png

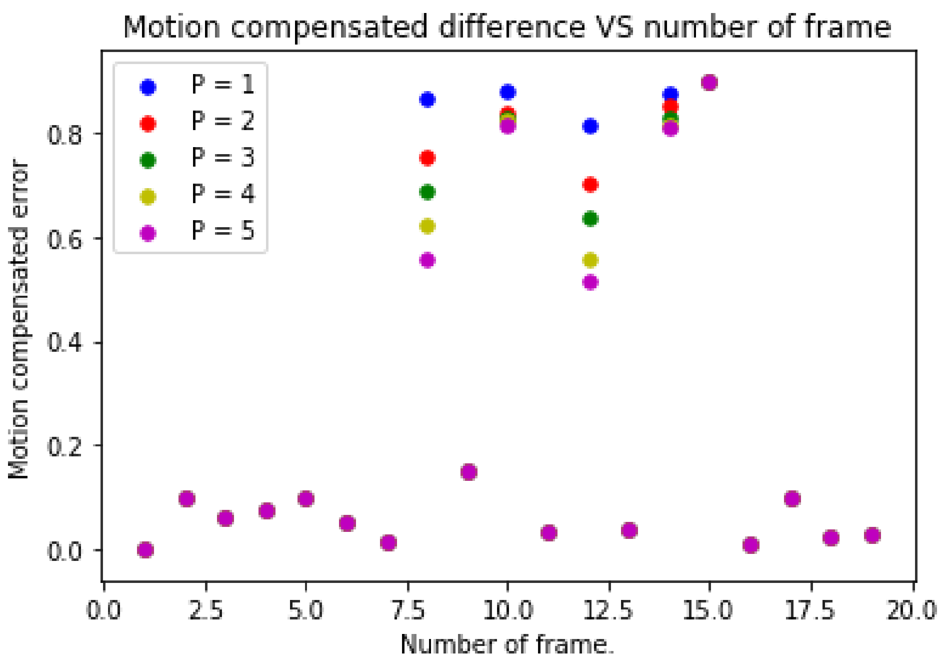


Figure 8: Motion compensated error by frame.

From the plot we can see that the error values are not too high. This is because the part we have chosen does not contain big movements on frames. We can also see that with higher values for p parameter the error is lower which makes sense because the best match has been seeked on a bigger surface.

The last study we have performed is the plotting of the motion compensated error vs the mae computed with the current and the reference frame, the motion estimation error:

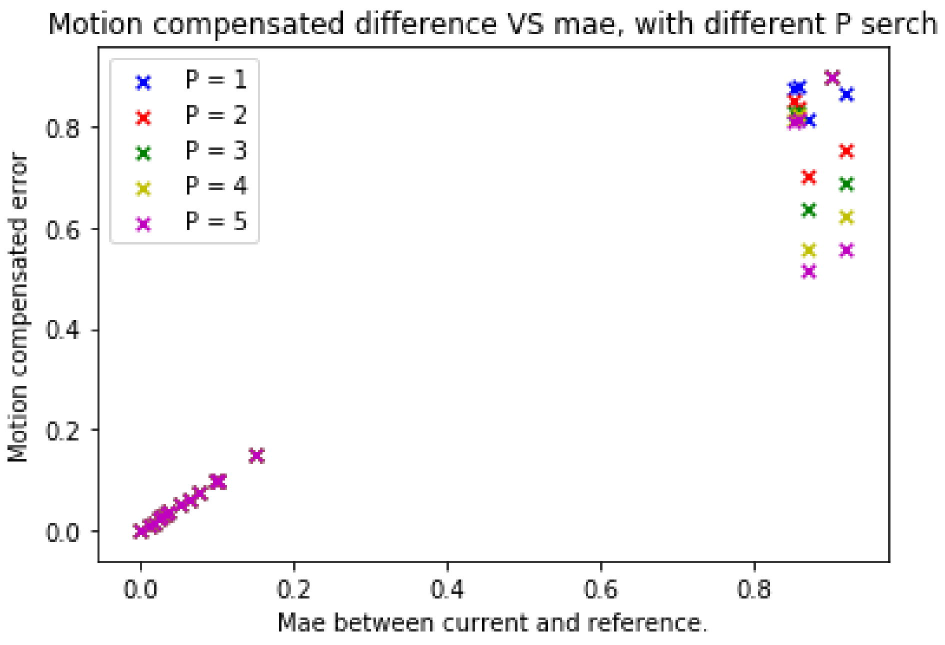


Figure 9: Motion compensated error by frame.

We can see that on left side of the plot where the errors are low they perform in a linear way and that there is not a big dependency with the p parameter. Also, it is clear than when there is more movement the error increases and that with a higher parameter of p the errors are lower, results that are totally consequent.

Another study that might have been performed is to study how things change with the grid parameter, m. We think that for higher values of m the processing times shrinks a lot but also the performance is lower with the consequent rise of the motion compensated error.

4 Lab 7

4.1 Introduction

During the last laboratories we have performed motion estimation and we have built up frames from a reference one storing the motion vectors and used them to move the pixels. The purpose of this lab session is the implementation of a video codec and the comparison of its performance with a MPEG standard video compression. The video has been trimmed from second 5 to second 6 and we are playing with 15 frames.

We will begin by building the video encoding pipeline of the page 4 of the lab. To do that, as we have been doing on the last labs, we will determine and reconstruct the E_{res} and the motion compensated frame. We will now determine the motion-compensated frame between the third one and the previous predicted second frame and we will store again the associated compressed E_{res} and we will predict the third frame in the same manner. After that, we will iterate through the previous points to predict all frames from our video and with all that we will reconstruct the frames in our video. To do so, we will follow up the next figure:

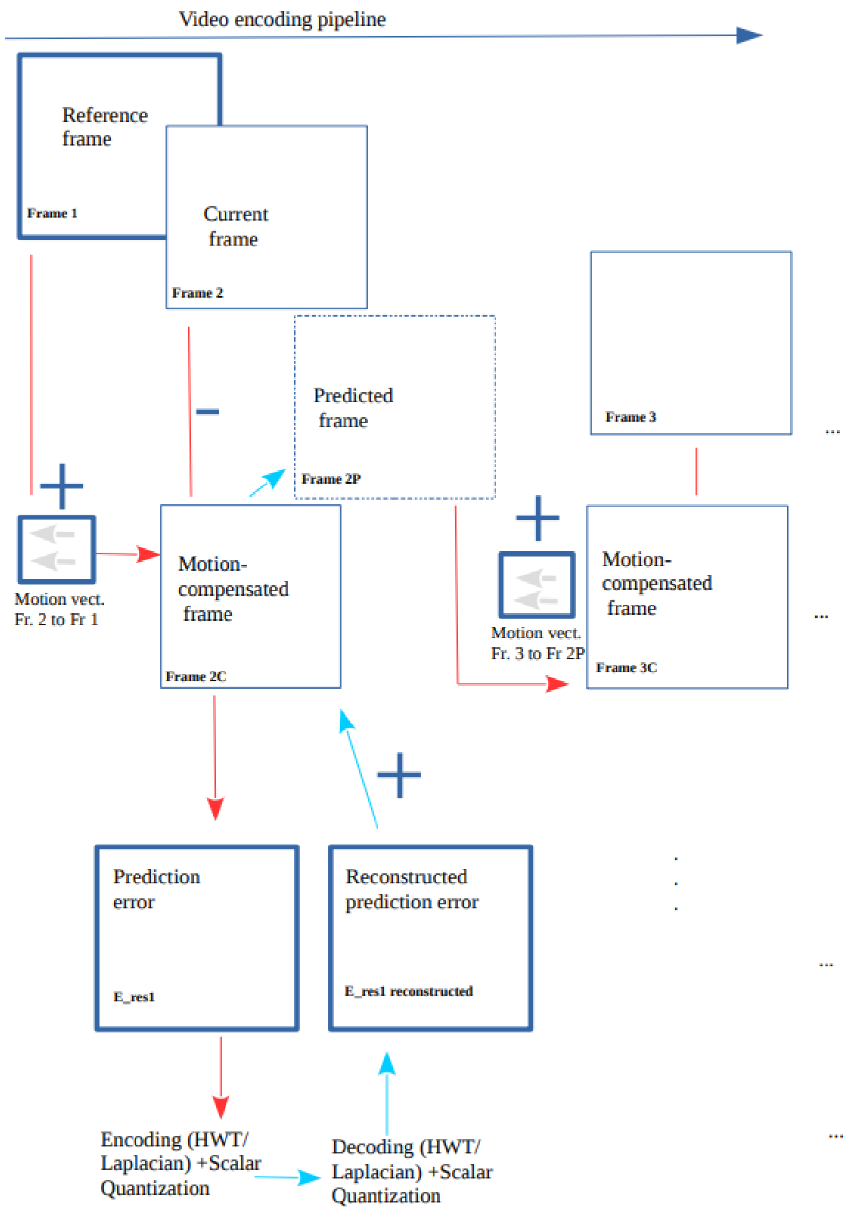
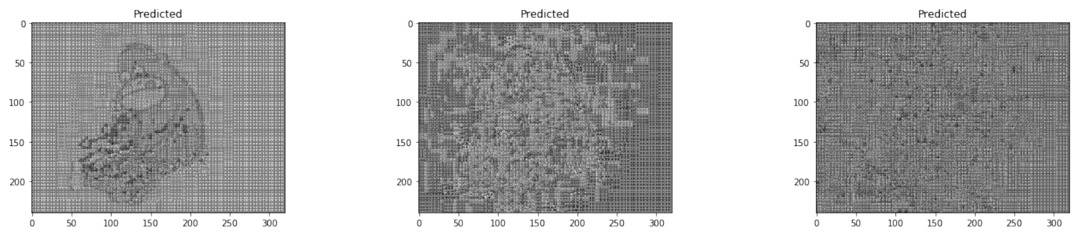


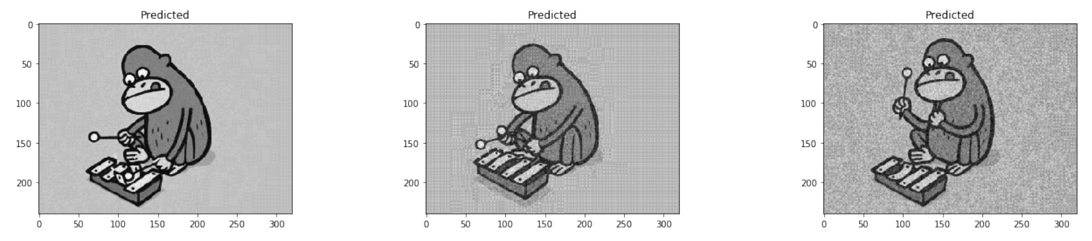
Figure 10: Motion compensated error by frame.

Since we are executing this process for the different values of R and we have error since the very first constructed frame we will propagate and add errors through the whole video construction. We can compare the same groups frames (4, 8 an 12) for different quantization levels (in this case levels 1, 4 and 8) and see what happens.



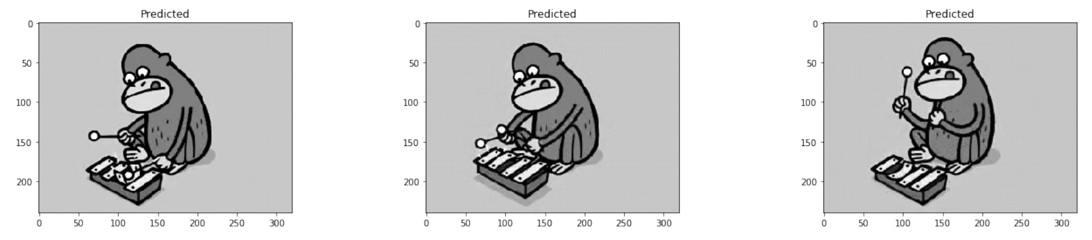
(a) Frame 4 (b) Frame 8 (c) Frame 12

Figure 11: Different frames for R = 1.



(a) Frame 4 (b) Frame 8 (c) Frame 12

Figure 12: Different frames for R = 4



(a) Frame 4 (b) Frame 8 (c) Frame 12

Figure 13: Different frames for R = 8

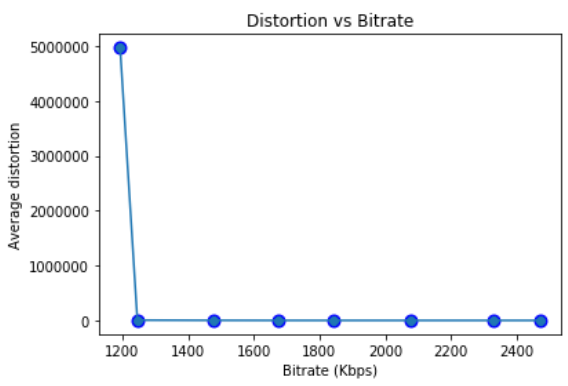
We can see that the figures are consequent with the results obtained on the previous laboratories. We observe distortion on the images and a bigger error propagation with lower levels of R. We infer rather high amount of errors since we are using frames with a lot of movement. With each iteration frame by frame and level by level we are computing and saving the values of distortion, average distortion and PSNR.

In order to measure the quality of compression of the videos and the size of information for each of them we will compute the video bitrate. The bitrate represents the ratio of information our screen is showing by second:

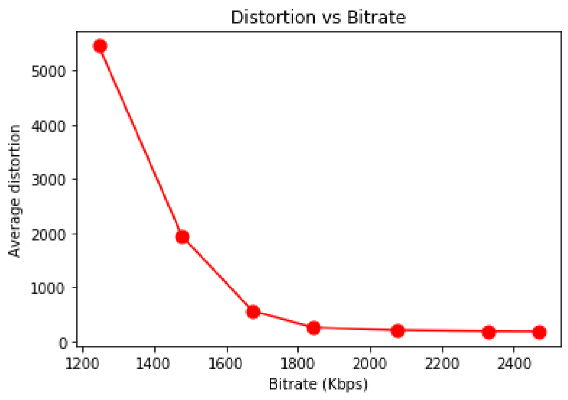
Listing 5: Building up of the motion compensated frame

```
R_video_bitrate = []
for i in R_huffman_list:
    bitrate = Frame_w * Frame_h * fps * i / 1024
    R_video_bitrate += [bitrate]
```

This video bitrate will be useful for plotting both the distortion and the PSNR vs the video bitrate:

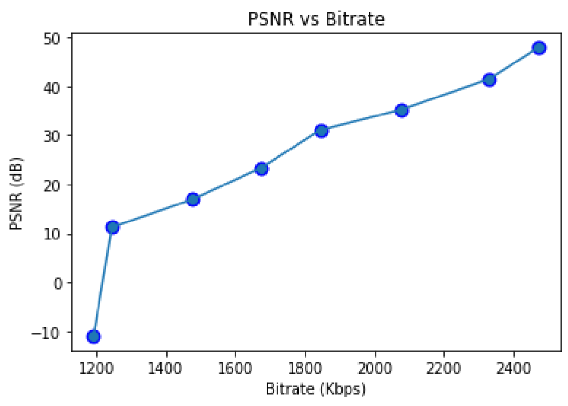


(a) Distortion vs bitrate for all levels

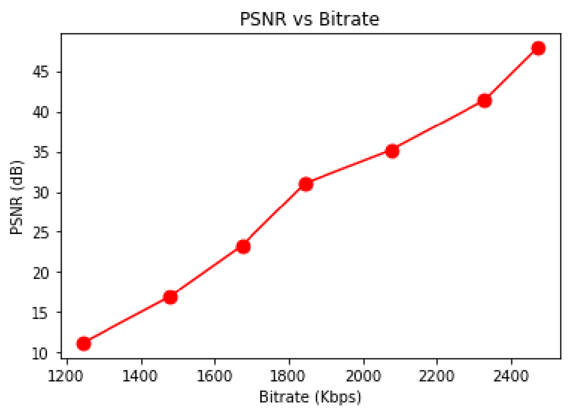


(b) Distortion vs bitrate for levels 2:8

Figure 14: Distortion vs bitrate (Kbps)



(a) PSNR vs bitrate for all levels



(b) PSNR vs bitrate for levels 2:8

Figure 15: PSNR (dB) vs bitrate (Kbps)

As we can see, in both graphs for the figure on the left there is something strange happening with the first level. The distortion is super high and so the PSNR is below zero. This might make sense because of the propagation of the huge errors we have on the first level and so the PSNR might be negative because we have logarithms of decimal values since the part of downside, the distortion is quite high.

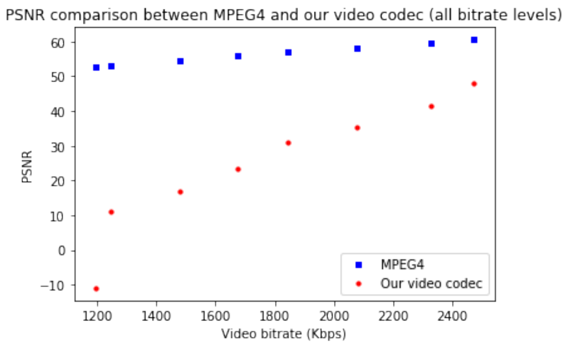
In any case, on the figures of the right hand side where we are omitting the first level are consequent with the theory. The more bitrate a video has, the less distortion for the video, the more information it carries and the more clear it will be printed. Actually, it suffers a kind of exponential decrease with a saturation value for R = 5 or 6 as we have seen on previous labs. For the PSNR, we can notice almost a lineal behaviour with the bitrate. Both results are also consistent with what we could see with our eyes from the printed images.

4.2 MPEG4 video coding standard comparison

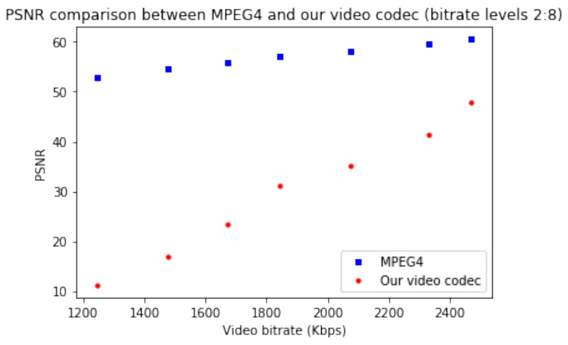
In order to write the video sequence in a MPEG-4 format, we have used the online video converter: (https://www.online-convert.com/). For this purpose, we have chosen the *.mp4* format for the different quality values determined by the bitrate calculated in the previous section for each possible value of R. As a result, we will obtain 8 videos with different video bitrates from the trimmed video.

Once we have all the videos in the correct MPEG format, we have computed the distortion and the PSNR for the different values of R (in this case determined by the video bitrate). If we compare those results with the ones obtained from our video codec, we get the following plots:

PSNR Comparison:



(a) PSNR comparison for all levels

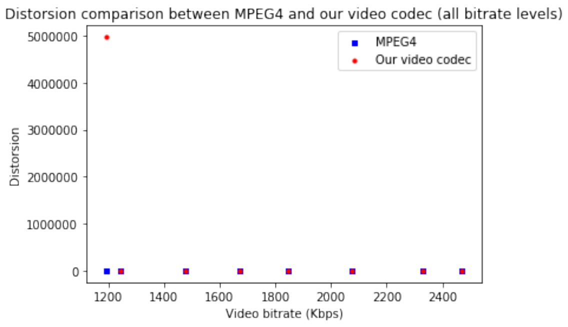


(b) PSNR comparison for levels 2:8

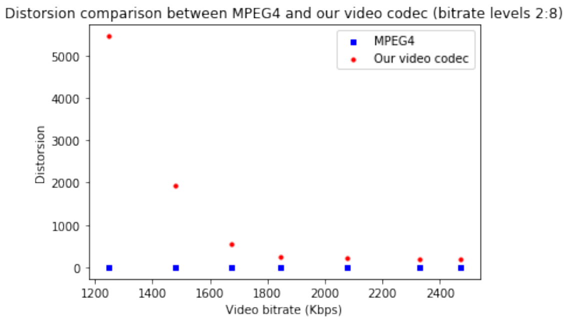
Figure 16: PSNR (dB) vs bitrate (Kbps). MPEG vs Our video codec

As stated in the previous section, if we analyze both plots we can see how, in our codec for the first level of R, the PSNR is very low. If we compare the PSNR between the levels 2:8, we can see that the MPEG4 codec has better quality than our video codec for all possible values of R. In both cases the PSNR increase linearly, but in the case of the MPEG codec remains more constant, while for our video codec increases in a more pronounced way.

Distortion Comparison:



(a) Distortion comparison for all levels



(b) Distortion comparison for levels 2:8

Figure 17: Distortion vs bitrate (Kbps). MPEG vs Our video codec

For the distortion, we obtain similar conclusions to the previous ones. For the MPEG4 codec the distortion remains practically the same for the different values of video bitrate, while for our codec, the distortion decrease when we increase the video bitrate. Our video codec distortion almost reaches the MPEG4 values for high values of video bitrate.

As general conclusions of this section, we can affirm that the MPEG4 codec is very optimized comparing with the codec that we have implemented during these last 3 laboratories.