Parvus - Developing a Video Codec

João Branquinho
76543
University of Aveiro
joaobranquinho@ua.pt

Luís Silva
76585
University of Aveiro
lfssilva@ua.pt

Tiago Ramalho
76718
University of Aveiro
t.ramalho22@ua.pt

***Abstract*—A study into the field of video compression. In it, we will explore some video compression techniques both lossless and lossy ones. We explain our approach to developing a video codec capable of doing this. There are some experimental results presented.**

I. Introduction

Nowadays video is everywhere, recent studies report that Netflix is responsible for 15 % of the Global Internet Traffic, followed closely by Youtube that hogs 11.4 %. This is a massive amount of data dedicated solely for video. For instance, if we imagine a 1080p video, this means that has a resolution of 1920 * 1080 this translates in 2073600 pixels per frame in a 30 fps video this would translate in 6220800 pixels to code each second, if each pixel we use 3 bytes to encode its value we would end up with a simple 5 min video taking up to 45 Gigabytes of data. This value would be outrageous, even with the fast modern internet there would be no way to handle this massive volume of data. This is the reason video compression is needed.

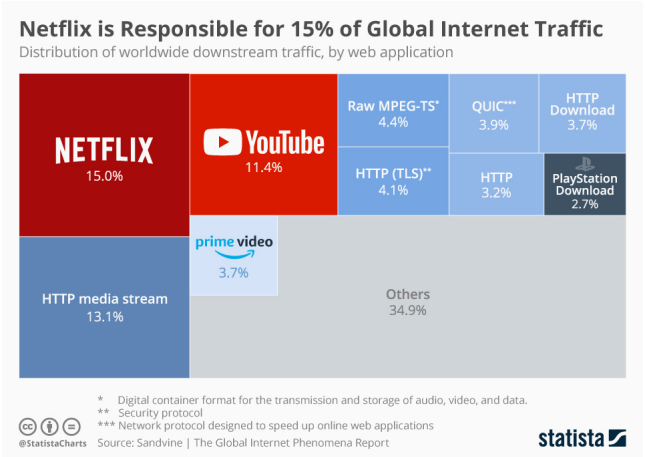

Fig. 1. Stats about global internet traffic

In previous work, we dwelled mainly with audio and audio compression. In this assignment, we implement a video codec, able to encode video sequences in a lossless and lossy manner.

We started by developing a video player that can play video sequences in different formats. Then the real developing of the video codec started. In this report, we will describe the steps and decisions we take while developing this codec.

II. Video Player

The first part of our assignment was to develop a video player able to play videos in different formats, namely different color spaces. In this section, we will talk a little about these different color spaces and about our program structure.

A. Color Spaces

The human eye is responsible for the human's vision. He does so by capturing electromagnetic radiation in a specific range of wavelengths. This range is called of the visible spectrum, usually, this compresses the wavelength between 400 and 700 nm. The characteristics that enable a color to be distinguished are the brightness, the hue, and the saturation. A digital image is compressed of pixels. Usually, these pixels are displayed in a screen where each pixel is expected to be described by its components of the color.

Based on this several color models emerged, called color spaces, trying to abstract and emulate in a digital system these different aspects.

One of those models is the RGB (Red Green Blue), specifically the $sRGB$ (Standard RGB. This model is the default for digital imagery especially if each pixel is coded with three different values, the Red, Green, and Blue value.

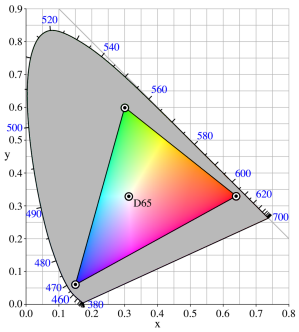

Fig. 2. Color Triagle defined by sRGB primary colors

Another important color space is the YUV/YC_bC_r. This color space is based on the color properties discussed above. Gives predominance to the Y component it corresponds mainly to the green zone which is the area our eyes are more sensitive to. The other components correspond to the blue and red respectively. These color spaces are interchangeable, meaning we can easily pass values from one to the other.

Namely to convert YUV to RGB the following equations are used:

$$R = 1.164 * (Y - 16) + 1.596 * (C_r - 128) \quad (1)$$

$$G = 1.164 * (Y - 16) - 0.813 * (C_b - 128) - 0.391 * (C_r - 128) \quad (2)$$

$$B = 1.164 * (Y - 16) + 2.018 * (C_b - 128) \quad (3)$$

One last process that we think is important to explain is the one called chrominance subsampling. As we previously stated, the human eye is more sensitive to the greens which are represented by the Y component. One technique commonly used in video compression is the subsampling of the chrominance plane. Meaning that this planes will have less information than the Y plane. In this assignment we dwelled mainly with three main types of the YC_bC_r:

- **4:4:4** There is no chrominance subsampling.
- **4:2:2** For each value of chrominance there are two of Luminance
- **4:2:0** For each value of chrominance there are four of Luminance

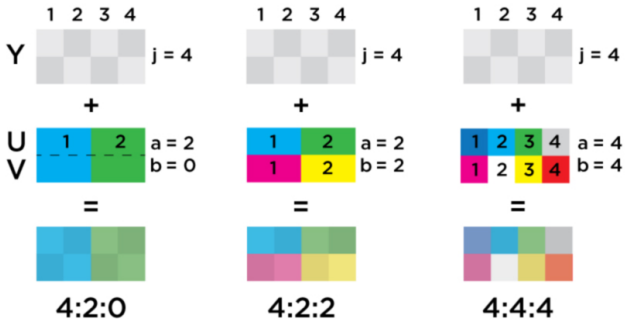

Fig. 3. Chroma subsampling visual depiction

B. Program Structure

In this segment, we will explain and explore the player structure. First, as suggested by the professors we started by creating a class frame that enables us to abstract the process of working with frames.

For storing frame information we harness the power of the openCV library that offers us the Mat object. With this object, we can create and easily manipulate Matrixes. From here could opt to store all the information in one single multi-layered matrix or divide this information in multiple matrices. We took the second option, meaning that, internally we would handle the frames in a planar way with a distinct matrix for each plan of information. With this decision, we were able to build classes that were able to inherit this Top class Frame based on the type we could manipulate the size of the chrominance matrix, maintaining the way with which we interact all the same throughout the different types possible. One of the abstractions that this Frame class enables is the conversion between color spaces that are easily made through a function call.

While developing the video player we noticed that we were not getting the framerate we desired, this was due to the overhead of conversion between the frame original format and the RGB needed for the video display.

What we opted to do was to use threads, meaning, while the program is running there are 2 threads working. One is responsible for retrieving the Frame information of the video File and decode that info into RGB it then puts it into a FIFO that connects the two working threads. The main thread is then responsible for only retrieve at the right moment an RGB image from the FIFO and display it on the screen. with this approach, we were able to get the Frame Rate we wanted.

III. Parvus Video Codec

The most important part of any new project is the name. The one we come up with was *Parvus* it comes from the Latin and it means small we tough that it made sense in every possible way.

The **Parvus.cpp** is the main of our codec. We have three possible modes, the lossless intra-frame codification, a lossless hybrid codification where we have intra-frame and inter-frame codification and a lossy codification using intra-frame and inter-frame codification too.

In this section we give a little overview of the codec capabilities alongside its use instructions. We also give a little explanation about the Golomb Code use.

A. Instructions

At any time the help command is available to show the possible run options.

```
$ parvus -h
```

There are three main modes that our codec can run:

- Lossless Intra Only
- Lossless Hybrid
- Lossy Mode

While in The first mode the only parameter needed is the mode.

```
$ parvus -f file.y4m -m 0
```

In the Lossless Hybrid Mode besides the mode there are other parameters needed:

```
-p, --periodicity arg  Periodicity
-b, --blocksize arg    Block Size
-s, --searcharea arg   Search Area Size
```

These will be explained in detailed in their respective section.

Still using the hybrid Mode we can go even further and quantized the residuals, for this we must specify the quantization steps we want for each plan

```
-y, --shamntY arg      quantization step Y
-u, --shamntU arg      quantization step U
-v, --shamntV arg      quantization step V
```

The last mode is our very own MPEG mode, for this one you should pass also the periodicity, blocksize, search area and change the mode to 2.

Finnaly, for decoding, you only need to pass the decode flag and the mode in each it was encoded, every other argument goes in the bitstream.

B. Golomb

In all modes, we need to generate the smallest bitstream we can. This new file has all the information for the decoder revert the coding and generate a video file. The bitstream has a lot of information like the mode who create that file, the type of the coding frame (intra or inter), the residuals, the motion vectors, and other pertinent data. With the purpose of using the minimum bits possible for writing the previous values, we use Golomb coding.

The **K** used by this optimal prefix code isn't always the same and depends on the values to encode. For each value to be encoded we calculate the real value which will be encoded:

```
if(value_to_encode >= 0)
   to_calculate_k += value_to_encode * 2;
else
   to_calculate_k += -2*value_to_encode-1;
```

after this transformation we send the **to_calculate_k** and the number of values for the function **get_best_k** and as the name refers the return is the best **K** for that values. This K needs to be on the bitstream too.

IV. Lossless Intra Frame Coding

For this phase, there are several things that we need to take into consideration. First of all, lossless encoding means that we cannot afford to lose any piece of information. For this, we need to take advantage of redundancy in order to save on some precious bits. When retrieved, the information must be exactly like the original file. The second aspect is that this method is an Intra Frame only coding method, this indicates that we are only taking advantage of redundancy from within the same frame. It's said that intraframe coding takes advantage of special redundancy. One way we could look at it is that it is almost like encoding every frame of the video as if they were individual images.

Our approach has two main moments. One is the prediction and the other is the efficient encoding of the residuals. For the prediction phase, we use the non-linear predictor of the JPEG-LS the LOCO predictor.

The residuals that are coded are simply the difference between the predicted value and the real one. This means that while decoding the decoder can simply predict the pixel value the same way the encoder did and then simply add the residual difference to obtain the original value. The reason this results in a reduction of the number of pixels is simply based on the efficient encoding of the residuals.

With this histogram, we can see that the values are distributed geometrically around zero. This means they can be efficiently encoded with the Golomb code.

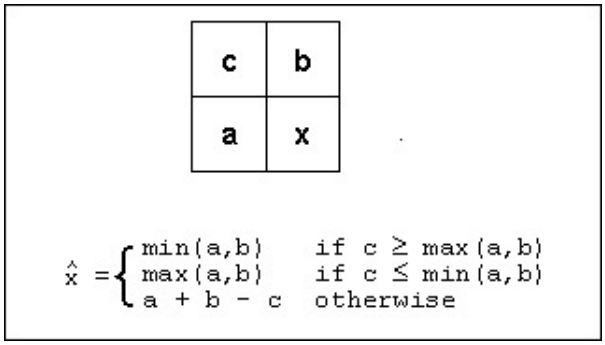

Fig. 4. LOCO predictor

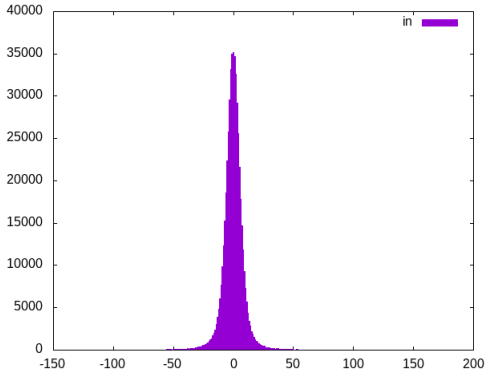

Fig. 5. Residuals of LOCO prediction histogram

This predictor embodies a primitive edge detection mechanism where vertical and horizontal edges can be identified by verifying the neighboring pixels.

There is only one problem with this approach, the first row. We don't have the necessary values to predict the first row so what we end up doing is to predict linearly the first values, meaning we predict the next value is the same of the pixel before it.

V. Lossless Hybrid Coding

The objective of a lossy hybrid video encoder is to contemplate 2 types of frames:

- Intra frames (Type I)
- Inter frames (Type P)

In section IV we explained the how we code the Type I frames. Interframes are encoded in a different way, as explained later. Before explaining our implementation of interframes there is another concept that requires the reader's attention: Motion compensation.

Motion compensation is a technique used in Type P frames. The idea behind it is that frames are not completely decorrelated. Frame p can only differ of frame $p-1$ in the dislocation of a single object and the remaining frame suffers no change (figure 6). This poses as a compression opportunity.

Our implementation of the interframe relies on the existence of some extra parameters: a searching area, a n which is the

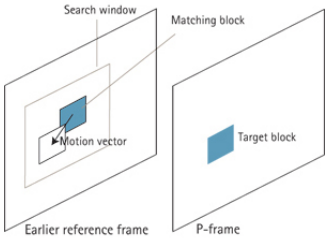

Fig. 6. Motion compensation

size of the side of the macroblock and a periodicity. Based on this parameters we perform the following steps:

1) We divide the component Y of the frames in macroblocks of $n * n$ size.
2) For each macroblock, we search for the most accurate representation of the macroblock around it in the predecessor frame (component Y).
3) After finding the best match we calculate the residuals.
4) We encode both the vectors and the macroblock.

The search for the best match is performed in a limited area around the macroblock. This area is established by the searching area parameter. The best block is calculated using a function provided by OpenCV called matchTemplate. One must notice that the chrominance components of the frame are calculated based on the calculated vector. For instance residuals for the U component are the subtraction of the U macroblock's component and the U component of the macroblock to which the previously calculated pointer points.

A curious feature of our work is the Golomb encoding used. For each frame's component there is a Golomb M. For all of the existing vectors there is another M.

VI. Lossy Coding

Some cases the using lossless coders can't compress enough the file and is needed some loss of information to reach the goal. In the following sections, we will explain our lossy encoders.

A. Quantization

Our first approach was based on the quantization of the residuals. So we add to the previous stage just this new feature. If the user included in the command the flags **-y -u -v** we use the respective numbers to quantify the residuals of the different components. For this simpler approach, the decoder needs to read the values of the quantization and do the inverse operation of the shift in the encoder.

VII. PARVEG (Lossy Coding based on MPEG)

One of the most well-known methods of lossy audio/video encoding is MPEG-1. This standard has become the most widely compatible lossy audio/video formats in the world and is used in a large number of products and technologies. The video compression component of MPEG-1 is heavily influenced by H.261. With this in mind, our group decided to develop an encoding method, the *PARVEG*, which strongly resembles some of the key features of both the MPEG format and consequently the *JPEG* format.

PARVEG is composed of two types of frames:

- The intraframes; and
- The non-intraframes.

A. Intraframes

Intraframes of PARVEG are comprised of a set of key features inherited from jpeg. Below we outline the steps used in order o encode an intraframe:

1) The frame is divided into macroblocks of 8x8 pixels.
2) DCT is applied to the matrix, creating a map of frequency components.
3) Each of the macroblocks is divided (per-element division) by a quantization frequency coefficient.
4) The resulting matrix is encoded using a combination of RLE and Golomb coding.

1) DCT: It is known that pixels in 2-dimensional images are correlated. The Discrete Cosine Transform, also known as DCT, takes advantage of that by expressing a sequence of data points, in our case an 8 by 8 macroblock, in terms of a sum of cosine functions. The basic DCT formula is the following (later on we will present a more efficient version of this formula):

$$G_{ij} = \frac{1}{\sqrt{2n}} C_i C_j \sum_{x=0}^{n-1} \sum_{y=0}^{n-1} p_{xy} \tag{4}$$

$$* \cos\left(\frac{(2y+1)\, j\pi}{2n}\right) \cos\left(\frac{(2x+1)\, i\pi}{2n}\right) \tag{5}$$

$$\text{for } 0 \leq i, j \leq n-1 \text{ and with } C_f = \begin{cases} \frac{2}{\sqrt{n}}, & f = 0 \\ \sqrt{2/n}, & f > 0 \end{cases}$$

By applying the DCT, the resulting macroblock concentrates the important information in the top left corner and becomes diluted when approaching to the bottom right one. This method relies on the low importance of high frequencies and how they can be discarded. The previously presented formula can be efficiency improved. After some algebraic enhancements, DCT can be expressed by:

$$G_{ij} = CPC^T \text{ with } C_{ij} = \begin{cases} \frac{1}{\sqrt{8}}, & i = 0, \\ \frac{1}{2} \cos\left(\frac{(2j+1)i\pi}{16}\right), & i > 0, \end{cases}$$

In our case, since we use OpenCV there is no need for implementation. OpenCV already provides an implementation one.

2) Quantization Coefficients: The application of quantization methods is where de information loss occurs. Each number of the DCT coefficients Matrix is divided by the corresponding number used in the quantization table. The JPEG standard allows the user to define the quantization tables to use in order for the user to try to archive maximum compression and image quality ratio. Obviously, in practice, most of the users have both no knowledge nor experience to apply their

own quantization tables. Taking this into consideration, there are two approaches that have prevailed over the years of study and improvement of this method.

The first approach consists of 2 default tables applied to 1) the luminance and 2) chrominance image components respectively. These tables are a result of many experiments by the JPEG committee.

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \tag{6}$$

Equation 6: Luminance

$$\begin{bmatrix} 17 & 18 & 24 & 27 & 99 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 66 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{bmatrix} \tag{7}$$

Equation 7: Chrominance

It is easy to see that the coefficients values increase from the top left corner to the bottom right corner. This is the way JPEG reduces the DCT coefficients with high spatial value.

The second approach is a quantization table calculated based on the expression

$$Quant\,(i,j) = 1 + R\,(i+j)$$

The parameter R is user-defined and can be fine-tuned in order to archive a better compression/quality ratio. The PARVEG method only implements the first method in order to reduce both user entropy and complexity. Although a deeper study on both methods would be interesting.

3) Run-length encoding: The RLE is a very simple form of lossless data compression in which data is stored in the form of chained $(val, amount)$. This way of encoding is useful in very specific forms of data display: Where values repeat themselves consecutively. In our case, tables tend to have important information in the top-left corner and lots of zeros in the bottom right corner. The way we implement RLE is by iterating the macroblock's pixels in a zig zag way (As demonstrated in fig. 7)

B. Non-Intraframes

In order to encode frames using the non-intra mode, MPEG calculates the differences between the frame and its predecessor. And only then applies the JPEG. One could argue that applying the DCT to this difference is useless because the frame and its predecessor are decorrelated. This might be true or no depending on the video being coded, furthermore, DCT is also useful in this case given that it is followed by quantization.

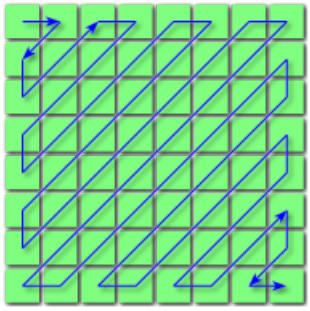

Fig. 7. ZigZag matrix travel

In our work, we had some set-backs and this feature, although implemented, is not working so all of the frames are intraframes.

VIII. Results

In Table I we can see our results of the parvus codec in lossless intra only. The experimental results translate very well what should be expected. First being this a lossless mode of encoding, the encoding rations are not very high. And second, the larger the file, the longer it takes to compress it. However, we can see that we have an average speed of alost 3,5 Mb per second.

In Table II and III we anayle the hybrid mode of our codec. We see that for the ducks' video we don't see much improvement. We could analyze and verify some obvious conclusions, like for instance, the increase in the searching area lead to an increase in the time it took to encode the video even if the resulted compression was not better. There is a plateau that is reached. Because of this results, we hypothesized that maybe the ducks' video was not the best to exemplify the new features, so we chose another video and in this one, we obtain far better results. In comparison, the new video was more still than the one of the ducks and this would make the motion compensation mare effective because there were fewer differences between takes only small moves.

In Table IV we could see some results of the lossy compression of the PARVEG.

Finnally, in Figure 8 and 9 we can see a side by side comparison of the variations of the Mean Squared Error and The Bitrate by frame. We can see that they are obviously closely related.

IX. Conclusion

The results of this study were very conclusive. Video compression is an indispensable mechanism in today digital environment, without it we would not be able to access easily to millions and millions of hours of video at any time we wanted because the overhead would be tremendous. With this

File Name	Original (M)	Compressed	Ratio	Time (min)	Time (s)	Speed (Kbits/sec)
in_to_tree_420_720p50	691	411	1.68	1	40	2800
in_to_tree_422_720p50	921	525	1.75	1	55	3443
in_to_tree_444_720p50	1400	764	1.83	2	58	3573
ducks_take_off_420_720p50	691	438	1.58	1	29	2843
ducks_take_off_422_720p50	921	581	1.59	1	50	3091
ducks_take_off_444_720p50	1400	862	1.62	2	47	3222
carphone_qcif	14	7.6	1.84	0	1.6	4000
tennis_sif	19	12	1.58	0	2.3	3043
stefan_sif	38	23	1.65	0	4.6	3261
akiyo_cif	45	18	2.5	0	5	5400
bridge_close_cif	304	178	1.71	0	36	3500
Average Encoding Ratio	1.76					
Average Encoding Speed (Kbits/sec)	3471					

TABLE I
CODEC RESULTS IN LOSSLESS INTRA ONLY MODE

File Name	ducks_take_off_420_720p50		Orignal Size	691			
Periodicity	Block Size	Search Area	Size	Ratio	Time (min)	Time(s)	Speed (Kbits/s)
10	8	4	445	1.55	2	12	1864
50	8	4	446	1.55	2	11	1870
100	8	4	446	1.55	2	18	1775
10	16	4	449	1.54	1	50	2200
10	8	8	443	1.56	2	43	1521
10	8	16	443	1.56	4	43	876

TABLE II
CODEC RESULTS IN HYBRID MODE FOR DUCKS FILE

File Name	akiyo_cif		Orignal Size	45			
Periodicity	Block Size	Search Area	Size	Ratio	Time (min)	Time(s)	Speed (Kbits/s)
10	8	4	11	4.09		9	3778
50	8	4	10	4.5		8	4375
100	8	4	10	4.5		7	5000
10	16	4	11	4.09		5	6800
50	16	4	10	4.5		5	7000
50	8	8	10	4.5		9	3889
50	8	16	10	4.5		17	2059

TABLE III
CODEC RESULTS IN HYBRID MODE FOR AKIYO FILE

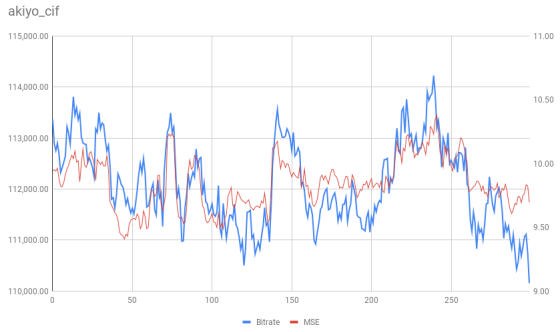

Fig. 8. Graph displaying The bitrate and the MSE of akiyo video

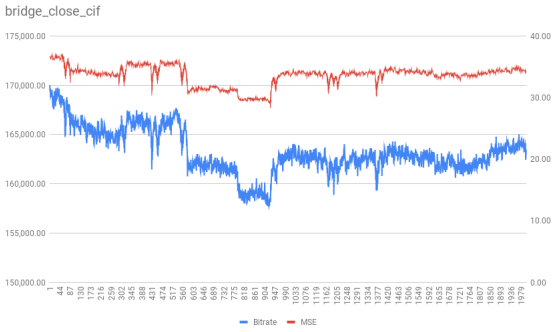

Fig. 9. Graph displaying The bitrate and the MSE of bridge close video

report, we experienced first hand the huge advantages a simple compression algorithm can make, whether your application can handle a lossy compression or not. By taking advantage of the flaws of our own sensory systems we can improve tremendously the amount of data we save. This was a great way to end an incredible journey throw the intricate world of data compression.

X. ACKNOWLEDGMENT

We would like to make a small acknowledgment to the groups of Ricardo Jesus, Inês Moreira and Ana Cruz for suggesting the use of threads in the video player.

File Name	Original (M)	Compressed	Ratio	Time (min)	Time (s)	Speed (Kbits/sec)	Time (min)	Time (s)	Average PSNR	Average RMSE
ducks_take_off_444_720p50	1400	196	7.14	4	8	4855	3	28	32.28	38.471738
in_to_tree_420_720p50	691	103	6.71	2	16	4324	1	41	34.16	24.969412
carphone_qcif	14	2.4	5.83		2	5800		2	34.07	25.470452
tennis_sif	19	3.4	5.59		3	5200		2.6	30.41	59.128798
stefan_sif	38	8.5	4.47		8	3688		5.3	30.58	56.883768
akiyo_cif	45	4.9	9.18		7.2	5569		6	38.2	9.848731
bridge_close_cif	304	48	6.33		56	4571		40	32.87	33.587977
Average Encoding Ratio	6.46									
Average Encoding Speed (Kbits/sec)	4858									
Average Average PSNR	33.22									
Average Average RMSE	36.88									

TABLE IV
CODEC RESULTS IN LOSSY WITH DCT

Moreover the authors would like to thank to their families for all the love and time wasted on raising them, to the teachers Armando Pinho and António Neves for their availability to answer questions.

REFERENCES

[1] M. J. Weinberger, G. Seroussi and G. Sapiro, "The LOCO-I lossless image compression algorithm: principles and standardization into JPEG-LS," in IEEE Transactions on Image Processing, vol. 9, no. 8, pp. 1309-1324, Aug. 2000. doi: 10.1109/83.855427