# Project 1

1st Bruno Mendes
Aveiro, Portugal
bmendes13@ua.pt

2nd Pedro Raimundo
Aveiro, Portugal
pedro.raimundo@ua.pt

3rd Rodrigo Silva
Aveiro, Portugal
silvarodrigo@ua.pt

***Abstract*—In this Document, we will give a brief explanation of our ideas for the resolution that we get for each exercise propose and what model we applied to resolve those problems.**

***Index Terms*—Compression, Final Context Model, Quantization, Uniform Scalar Quantization**

CONTENTS



## I. Introduction

This Document, we will talk about all the eleven exercises purpose by this Project 1, and explain in detail, how we did it and why we decided to go for that technology of manipulation of video and sound files.

## II. Exercise 1

In this first exercise, we were asked to copy a wave sample file to sample, so we used the python library called wave. What this library allows is that we can find fields needed to create a file with the same characteristics and then within a cycle copy the samples in order from the original file to the copy file. With this exercise, it was possible to verify that the information about the characteristics of the file is described at the beginning of this and that if we remove that part of the file we get a file with data raw.
To run this exercise you need to have:
*python3 1-copy-sample-by-sample.py WaveFile*

University of Aveiro.

## III. Exercise 2

Here, just as in the past exercise the idea was to copy a file to another file, but an image input would be used and we had to copy it pixel by pixel. The approach in this exercise was to use the OpenCV library. The idea was to read the image we want to copy, then create an image of the same size and copy every pixel of the original image of one position to the same position in the copied image.
To run this exercise you need to have:
*python3 2-copy-pixel-by-pixel.py ImageFile*

## IV. Exercise 3

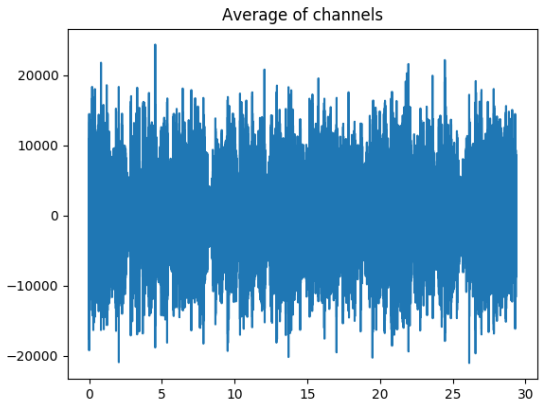
We were asked to implement a program that could display as much video as an image, so we used a library called MimeTypes, which with the name of the reading file we removed the information needed to know if it is a video or image. Depending on the result a different function was called, we both used OpenCV on the image/video read/display. In case of image, imShow of this library was used, and in case of being a video, VideoCapture was used and in a cycle, while there were frames to be displayed it would repeat until the video was over.
To run this exercise you need to have:
*python3 3-display-img-vid.py ImageFile or VideoFile*

## V. Exercise 4

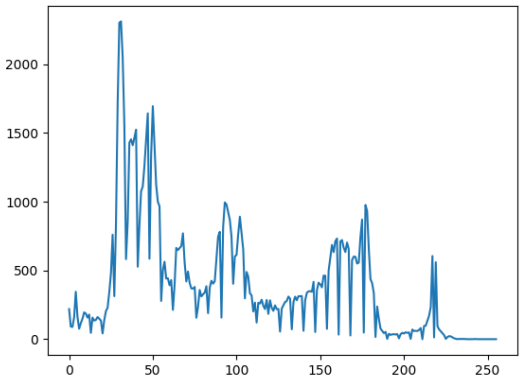
Here, it's asked to do a simple program that calculates an histogram of an audio file, even considering both audio channels. To understand it with both channels it's calculated its average between those two channels.
To start the program we need to use the following command:
python3 ex4.py "name-of-audio-file"

## VI. Exercise 5

The same principle is used here based on the last exercise, we need to calculate an histogram of an image or video file, even considering each color channel as well its gray-scale version. To show the color histograms, each histogram for each color is calculated using OpenCV, by using an index for each channel. To get the gray-scale version of it, we need to convert




it from RGB to GRAY and then show it on an histogram.
To start the program we need to use the following command: python3 ex5.py ”name-of-image-or-video-file”




## VII. Exercise 6

Applying the concept of uniform scalar quantization, we were tasked with developing a tool to perform said operation. In practice, assuming a transformation of an 8-bit sample to a 6-bit sample, the software will perform a bitwise right shift on each sample, effectively disregarding the 2 LSBs. The binary sequence is right padded with zeros, if needed.

One can easily see it running with the following command: python3 6-compression-audio.py input.wav output.wav

## VIII. Exercise 7

In this exercise, we were asked to implement a program that reduced the number of bits in an image or video. For that, we in each pixel made a reduction of the initial bit number to the number of bits that the user requested, and later we put back to the original color system. In this way, the colors were slightly distorted and the number of bits required to represent each pixel decreased.
To run this exercise you need to have:

Fig. 1. Original Image

Fig. 2. Resulted Image with 2 bits

*python3 07-VideoImageCompression.py ImageFile NewImage-File NumberOfBits*
**Results:**

## IX. Exercise 8

In this exercise we were given the task to calculate the SNR (Signal-to-Noise Ratio) as well its maximum sample absolute error on audio files.
To start the program we need to use the following command: python3 ex8.py ”name-of-audio-file-original” ”name-of-audio-file-encoded”
This program will read two bytes of each frame (each byte for each channel) between two audio files. It uses the following calculations:
SNR = 10 * log10(sum(original²)/sum(original² - encoded²))
If the SNR is 0, then there was no loss and the files are identical.

| FCM 2-bit seq | C(0) | C(1) | P(0) | P(1) |
|---|---|---|---|---|
| 00 | 4 | 3 | 57.14 | 42.86 |
| 01 | 4 | 4 | 50.0 | 50.0 |
| 10 | 3 | 5 | 37.5 | 62.5 |
| 11 | 4 | 1 | 80.0 | 20.0 |

Fig. 3. FCM order 2 result

## X. Exercise 9

This exercise uses the same principle we used on the last one, it's to calculate the SNR (Signal-to-Noise Ratio) as well its maximum sample absolute error on images and video files. To start the program we need to use the following command: python3 ex9.py "name-of-image-or-video-file-original" "name-of-image-or-video-file-encoded"
This program will read each frame of two images or videos and calculate their mean before comparing them. It uses the same following calculations:
SNR = 10 * log10(sum(original²)/sum(original² - encoded²))
If the SNR is 0, then there was no loss and the files are identical.

## XI. Exercise 10

The aim on this exercise was the implement a Finite-Context Model with parameterized order, which was fully implemented. It will begin by generating the possible binary sequence permutations and proceed to analyze the binary input file, bit by bit, *filling* the FCM table with 0 and 1 counters. As an example, assuming the following input parameters:

Input file: 00111000110101010101011000001100

FCM order: 2

The possible binary permutations of 2 would be 00, 01, 10 and 11. Reading the input binary sequence, we would get the following table:

## XII. Exercise 11

The objective of this exercise was the same as the previous exercise but using image or video files as input. This exercise was implemented just to a level of permutations. The idea behind this exercise was to go to all pixels and to each channel count the times that that byte appear if a determinate byte appear before that, then calculate the conditional probability, getting a table that contains all the probabilities of all events. Using that values we calculate the entropy and plot the results.
The entropy was calculated using the following formula:
Entropy = P(i) * $log_2$(P(i))
To run this exercise you need to have:
*python3 11-entropy-Video.py ImageFile or VideoFile*

**Results:**

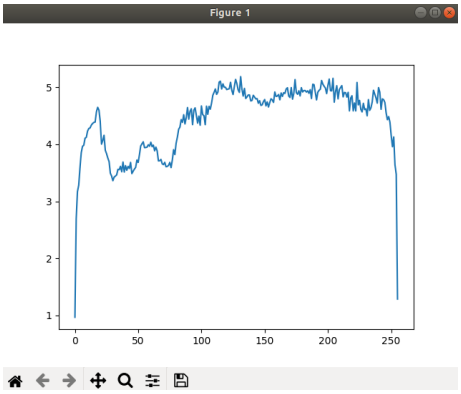



Fig. 4. Calculated Entropy for image

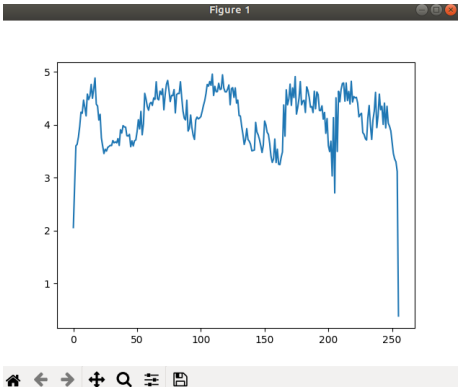



Fig. 5. Calculated Entropy for video