Universidade de Aveiro

Informação e Codificação
Projeto 1

Ana Filipe (93350), João Gameiro (93097), Pedro Abreu (93240)

Departamento de Electrónica, Telecomunicações e Informática

19 de novembro de 2021

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

O presente relatório visa descrever a resolução do Projecto 1 desenvolvido no âmbito da unidade curricular de Informação e Codificação.

O código desenvolvido para o projeto encontra-se disponível em: https://github.com/Torrakanor611/IC_project1.

Para este projeto foram utilizadas as seguintes bibliotecas : *OpenCV*, *AudioFile*, *libsndfile* e *matplotlib*. Os requisitos para a compilação estão definidos no ficheiro *README.md* no repositório.

No directório base do projeto existe um ficheiro *makefile* que tem como objectivo facilitar a compilação de todos os programas, para que não seja necessário a preocupação em inserir flags corretas. Seguidamente é apresentado um exemplo de como compilar um programa.

```
#Exemplo para compilar o exercício 5
#Estando no diretório base do projeto
make ex5

#Comando para correr o programa
bin/ex5 <program-arguments>

#Para compilar todos os programas
make all

#Para eliminar os executáveis
make clean
```

De notar que os executáveis de cada programa são guardados na pasta *bin*. Importante referir também que a pasta *files* possui recursos disponíveis para serem usados em conjunto com os vários programas.

Acrescentar também que foi realizado um exercício a mais pois pertencia ao guião antigo e foi resolvido antes da primeira actualização do enunciado (secção Exercício 4 video).

Capítulo 2

Parte B

2.1 Exercício 2

Para este exercício foi desenvolvido um programa que tem como argumentos o caminho para um ficheiro de texto de entrada e o caminho para um ficheiro de texto de saída. O ficheiro de entrada tem obrigatoriamente de existir enquanto que o ficheiro de saída não precisa necessariamente de existir (caso não exista é criado um novo ficheiro, caso exista é alterado o seu conteúdo).

Tendo o ficheiro de entrada, o programa vai copiar caractere a caractere o conteúdo do mesmo e escrever no ficheiro de saída. Para efetuar estas operações foi utilizada a biblioteca *fstream* que fornece ferramentas que permitem a leitura e escrita em ficheiros de texto.

A leitura de caracteres do ficheiro de entrada foi feita com recurso ao tipo de dados *ifstream* e a escrita com o tipo *ofstream*. Assim com recurso a um ciclo *while* foram lidos todos os caracteres até chegar ao *EOF - End of File*. Cada caractere lido era também escrito no ficheiro destino.

2.2 Exercício 3

Este programa tem como argumentos novamente o caminho para um ficheiro de entrada e o caminho para um de saída. No entanto neste caso ambos precisam de ser de áudio e o primeiro precisa obrigatoriamente de existir.

Nesta implementação foi utilizada a biblioteca *Audiofile*. Para a utilizar foi necessário descarregar do Github (https://github.com/adamstark/AudioFile) o ficheiro *Audiofile.h* e fazer a sua inclusão no cabeçalho do programa *ex3.cpp*.

A biblioteca permitiu criar um objeto do tipo *AudioFile* e a partir dos seus métodos ler e aceder à meta-informação de um ficheiro de áudio (após fazer a operação de *load* em que se especificava qual o ficheiro a aceder). De entre a informação importante neste tipo de dados é de realçar os seguintes exemplos:

- NumChannels -> que representa o número de canais num ficheiro de áudio;
- NumSamplesPerChannel -> que representa o número de samples por canal;
- BitDeph -> que representa o número de bits por sample.

Logo, para copiar um ficheiro sample a sample é preciso primeiro criar o futuro ficheiro output (representado no programa por um objeto *AudioFile*). Para isso são especificadas as suas características como o número de canais, de samples, o tamanho do seu buffer (que é igual ao número de canais vezes o número de samples por canal), a taxa de sampling e o seu BitDeph. Todos estes atributos são iguais aos presentes no ficheiro de áudio de entrada.

Estando o objeto *AudioFile* output pronto para receber a informação, falta apenas iterar sobre todas as samples do ficheiro origem para as copiar para o novo. Por isso são criados dois ciclos *for* um para iterar sobre cada canal e "dentro"de cada canal, outro ciclo para iterar sobre as suas samples. O valor de todas as samples é posto num buffer temporário cujo conteúdo é depois atribuído ao objecto *AudioFile* que representa o futuro ficheiro de output. Quando todo o processo tiver terminado, a informação recolhida é guardada no novo ficheiro de áudio.

2.3 Exercício 4

Para este exercício foi usada a biblioteca *OpenCV*, para abrir e aceder aos pixeis de uma imagem. Tal como nos exercícios anteriores, o programa recebe como argumentos os caminhos para duas imagens e o primeiro tem de obrigatoriamente de existir.

Relativamente à implementação, são criados dois objecto do tipo *Mat* sendo que um representa a imagem original e o outro a imagem output. O objecto *Mat* output vai ter as mesmas configurações que estão presentes na imagem original.

Após a criação dos objectos é feita uma iteração sobre todos os pixeis da imagem original. Finalmente com auxílio do tipo de dados *Vec3b* (vector com entradas de 3 bytes para representar os pixeis), são copiados os valores dos pixeis presentes no objecto *Mat* original para o objecto *Mat* output.

Após a cópia de informação, a imagem resultado é guardada. Antes do seu término, o programa ainda apresenta ao utilizador a imagem output numa janela auxiliar.

2.4 Exercício 4 video

Este exercício representa uma extensão do anterior, mas para vídeo e foi realizado antes da atualização do guião.

Assim sendo, este programa aceita como entrada o *path* para um ficheiro de vídeo e copia *frame by frame* para outro ficheiro fornecido no segundo argumento. O programa também faz o display dos frames com 30 milissegundos de intervalo, sendo que o utlizador pode forçar a sua paragem carregando na tecla 'q' ou ESC. Foi usada a classe *VideoWriter built-ined*

OpenCV. Esta classe permite facilmente, por exemplo, guardar num ficheiro o resultado do processamento de imagem de um trecho de vídeo.

O programa desconsidera o codec do vídeo de entrada, mas garante que o vídeo de saída é comprimido com o codec H.264. A definição do construtor do objeto *VideoWriter* di vídeo de saída é a seguinte:

```
VideoWriter vid_out(argv[2], VideoWriter::fourcc('a','v','c','1'), fps, frame_size);
```

[1] *fourcc* significa *four character code* e é um pequeno código de 4 dígitos que identifica o *codec* de video, formatos de compressão, de cores ou pixeis. No nosso sistema operativo o *fourcc* que representa o *codec H.264* é identificado pelas letras *a v c 1*.

Capítulo 3

Parte C

3.1 Exercício 5

O exercício 5 teve a sua base no exercício 2, pois a leitura dos caracteres de um ficheiro de input é feita da mesma maneira e com recurso ao tipo de dados *ifstream* da biblioteca *fstream*. Para além da leitura dos caracteres é contado também o número de ocorrências de cada um. Essas ocorrências são apresentadas num histograma que foi feito com recurso à biblioteca *matplotlib*.

Para a contagem do número de ocorrências de cada caractere usou-se a estrutura de dados *map*. Criou-se um mapa em que as chaves (tipo *char*) correspondiam aos caracteres do ficheiro de texto, e o valor associado a cada chave (tipo *int*) representava o número de ocorrências de cada caractere. Por cada caractere lido era verificado se o mesmo já estava presente no mapa e se não estivesse era adicionado ao conjunto de chaves e o seu valor correspondente era posto a 1. Caso estivesse presente no mapa, o seu valor correspondente era incrementado uma unidade.

Finalmente faltava calcular a entropia. Para esse cálculo era então necessário perceber o que era informação e como ela se inseria no cálculo da fórmula da entropia.

De acordo com Shannon, informação está associado à probabilidade de certos eventos acontecerem. Sendo assim, neste caso particular, considerámos os eventos como a ocorrência de símbolos no ficheiro. Para calcular a probabilidade da ocorrência cada símbolo dividiu-se o número de ocorrências do mesmo, pelo número total de caracteres no ficheiro. Tendo a probabilidade de cada símbolo, para calcular a informação usou-se a seguinte fórmula:

$$i(E) = -\log P(E)$$

Tendo a informação faltava apenas calcular a entropia que no fundo representa a informação média por símbolo. Para isso usou-se a fórmula:

$$H = \sum_j P(E_j) \cdot i(E_j) = -\sum_j P(E_j) \cdot \log P(E_j)$$

Analisando a tabela 3.1 podemos tirar conclusões relativamente à entropia. O ficheiro t.txt tem apenas dois caracteres diferentes. A sua entropia é muito menor que a do ficheiro lusiadas.txt que possui um número bastante elevado de caracteres diferentes. Podemos então afirmar que o ficheiro t.txt é muito mais redudante que o lusiadas.txt e que seria mais compressível. Analisando resultados para outros ficheiros chegamos então à conclusão que quanto maior for a entropia, maior a disparidade de informação e consequentemente menor e mais difícil será a compressão.

Ficheiro	**Entropia Correspondente**
lusiadas.txt	3.13889
t.txt	0.665785

Tabela 3.1: Ficheiros de texto e a sua entropia

Finalmente faltava apenas apresentar a informação num histograma. Para essa apresentação foi usada a biblioteca *matplotlib* e foi usado o método *bar()*. Este método para apresentar informação apenas suporta vectores de valores numéricos. Por isso foram utilizados 3 vectores auxiliares para apresentar correctamente informação no histograma.

- vector de inteiros *nums* -> que tem como conteúdo números de 0 até (número de diferentes caracteres).
- vector de caracteres *letters* -> Armazena os diferentes caracteres presentes no ficheiro texto
- vector de inteiros *values* -> Armazena o número de ocorrências de cada caractere.

Através da chamada à função *xticks* que recebe como argumentos os vectores *nums* e *letters* é possível efectuar o mapeamento dos valores que vão aparecer no eixo das abcissas (ou seja por cada valor presente em *nums* vai aparecer uma letra de *letters*). No fim faltou apenas definir que o histograma iria apresentar os valores de *values* em função do que é apresentado no eixo das abcissas.

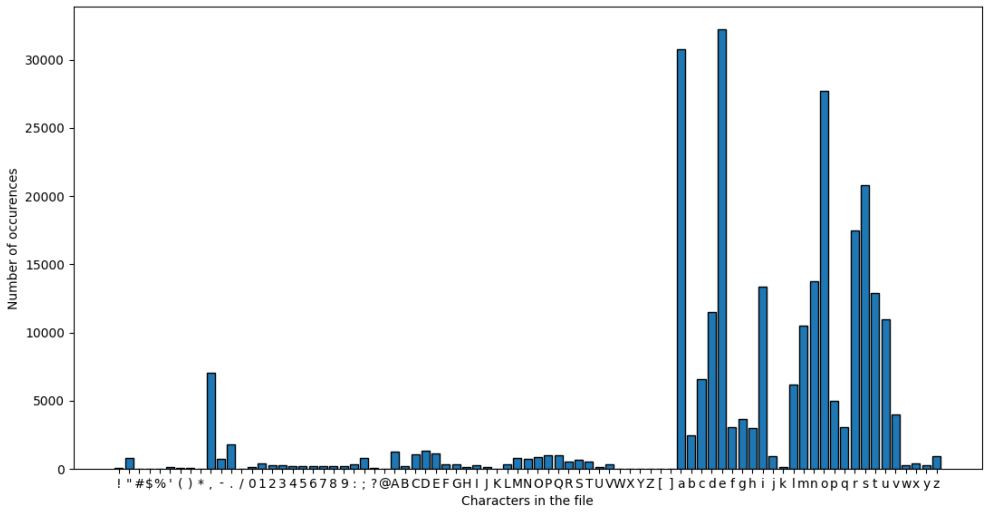


Figure 3.1: Resultado da execução do programa 5 com o ficheiro lusiadas.txt

3.2 Exercício 6

Este programa tem como argumento o caminho para um ficheiro de entrada, sendo necessário este existir e ser um ficheiro áudio.

Nesta implementação foi utilizada a biblioteca *libsndfile*. Com recurso a esta biblioteca foi possível guardar a informação num vetor, em que os índices pares referem-se ao primeiro canal e os índices ímpares ao segundo canal.

A contagem é feita com recurso a estrutura de dados *map*. Criou-se um mapa em que as chaves (tipo *short*) correspondiam aos valores lidos do ficheiro de áudio e o valor associado a cada chave (tipo *int*) representava o número de ocorrências desses valores. Usando estes valores é calculado a entropia com recurso à formula apresentada na secção 3.1.

Para o teste da Figura 3.2, obtivemos $H = 13.9086$ bits.

Para finalizar este exercício, faltava apenas apresentar a informação num histograma. Para este fim, foi usada a biblioteca *matplotlib* e foi usado o método *bar()*. No eixo das abcissas estão representados todos os valores das samples e no eixo das coordenadas está representado o número de ocorrências.

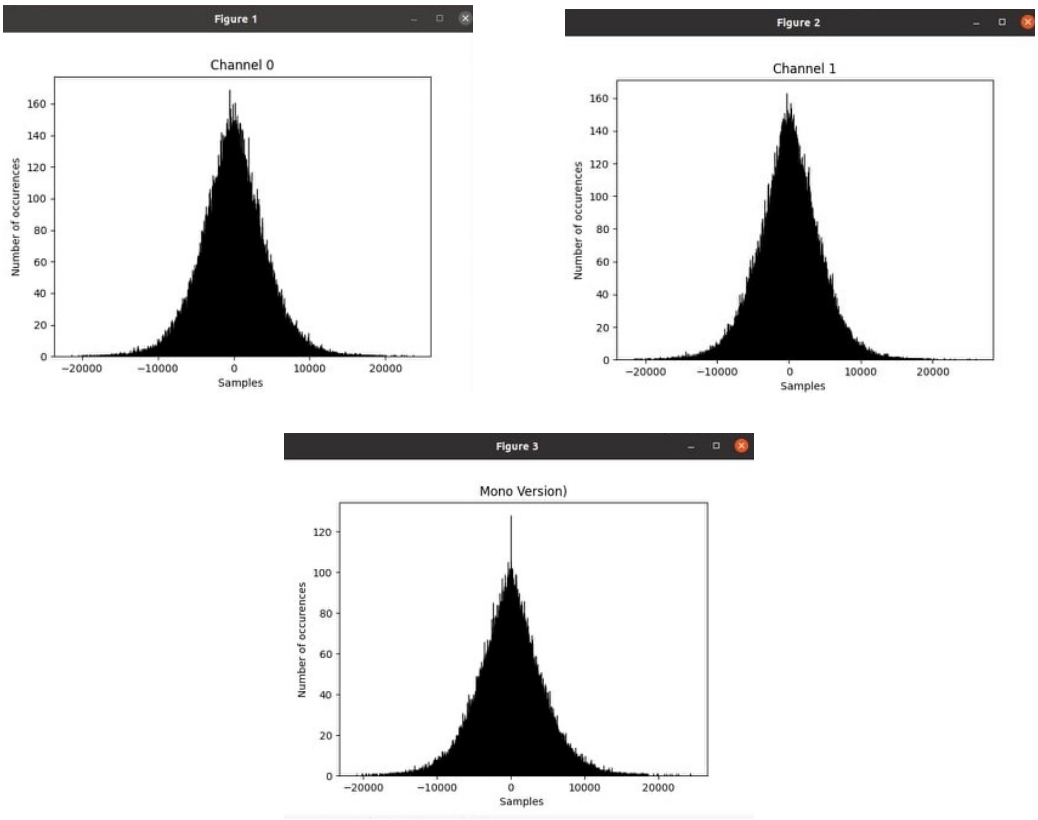


Figure 3.2: Resultado da execução do programa 6 com o ficheiro sample01.wav

3.3 Exercício 7

Este programa tem como entrada o *path* para uma imagem e tem como objectivo o cálculo do seu histograma e da sua entropia. São apresentados dois histogramas sendo que um é relativo ao espaço de cores RGB, o outro é em relação à imagem em escalas de cinzento. À saída o programa apresenta na consola a entropia da conjunta dos 3 canais de cor e a redundância.

Considerando que o alfabeto de símbolos é $\sum = \{0, 1, 2, ..., 255\}$ e assumindo *BitDepth* de 8, a probabilidade de cada símbolo é a divisão do número de ocorrências do símbolo nos 3 canais sobre o numero total de símbolos (pixeis * 3 canais).

Para calcular a entropia foi utilizada a fórmula referida na secção 3.1 mas aplicada ao caso especifico de imagens.

Foi calculado também a redundância, que representa a diferença entre a medida combinatória e a probabilística.

$$R = log\left|\sum\right| - H$$

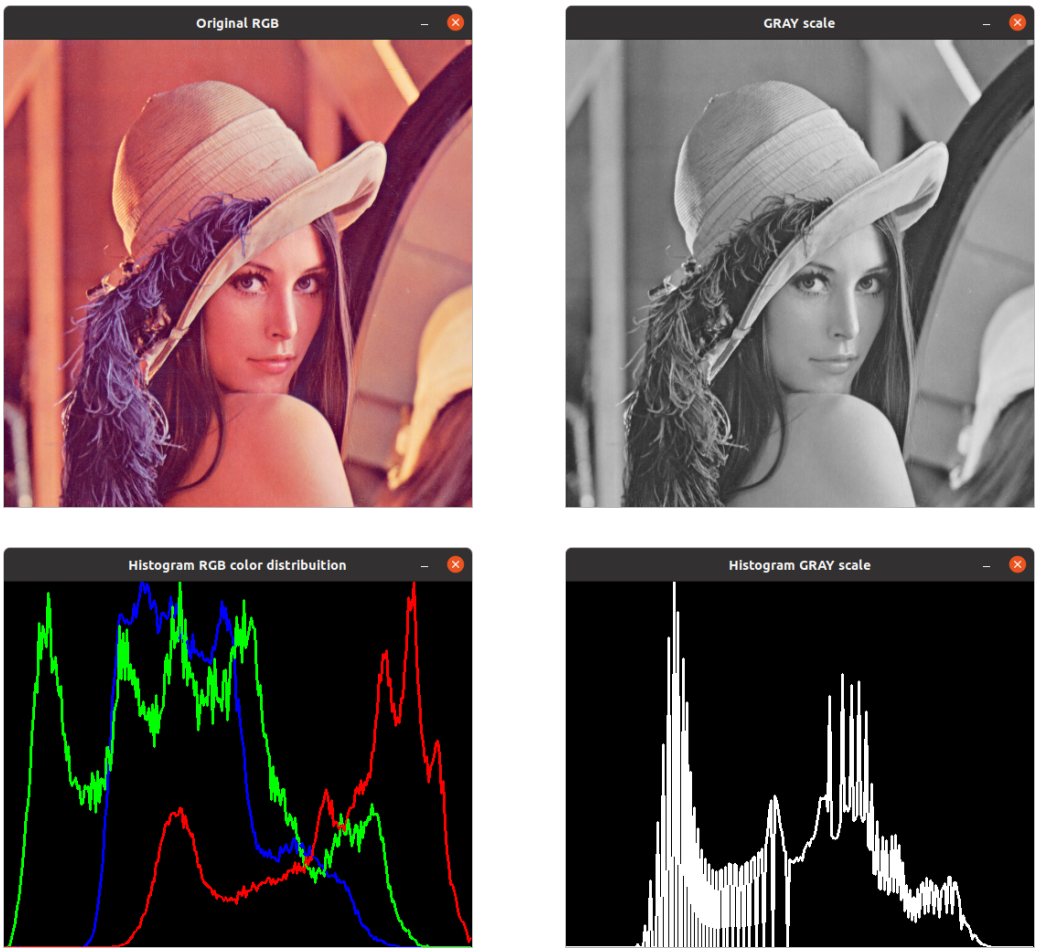


Figure 3.3: ./bin/ex7 files/miuda.png

Para o teste da Figura 3.3, obtivemos $H = 7.748378$ bits e $R = 0.251622$.

Para obter os histogramas dos canais de cores e da escala de cinzento usamos o módulo de histogramas do *OpenCV*. No eixo das abcissas estão representados todos os valores possíveis de cor em *RGB*.

Capítulo 4

Parte D

4.1 Exercício 8

Para este exercício era necessário reduzir o número de bits usado para representar cada sample de um ficheiro de áudio. Para o fazer foi necessário analisar o que é quantização e como fazê-la.

Em termos de processamento de sinais, o processo de quantização representa a transformação de um sinal contínuo num sinal discreto. Esta técnica de compressão de informação é irreversível, pois implica perda de informação que é irrecuperável se tentarmos reconstruir o sinal original. Para realizar a quantização às samples do ficheiro de áudio de entrada, foram então considerados dois tipos de quantização [2]:

- *Mid-riser* - A origem assenta no meio de uma "subida de um degrau"num gráfico tipo escada.
- *Mid-tread* - A origem assenta no meio do "topo de um degrau"num gráfico tipo escada.

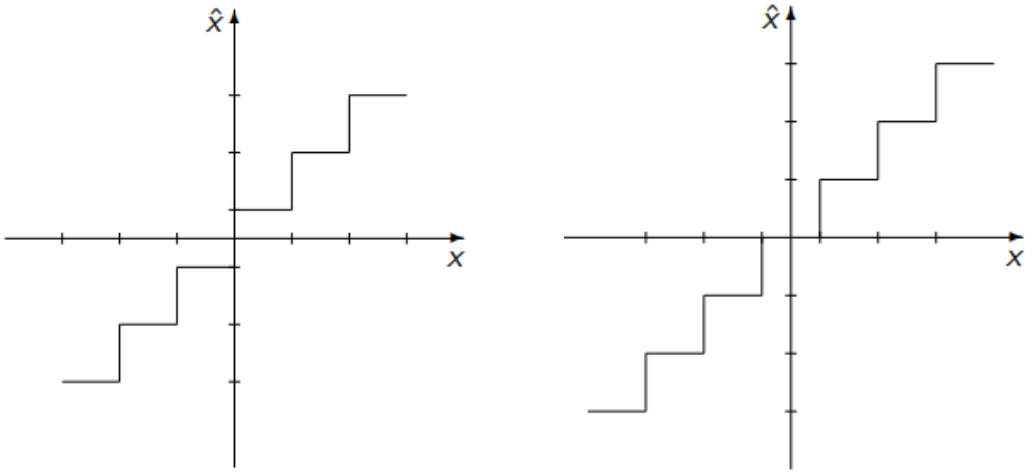


Figure 4.1: Referencia gráfica dos métodos de quantização Mid-riser(à esquerda) e Mid-tread(à direita)

Para compressão de vídeo e imagem é comum favorecer-se algoritmos como o mid-tread pois este preserva o nível 0 no processo de quantização.

Deste modo, o programa que implementa o processo descrito anteriormente, tem a funcionalidade adicional de questionar o utilizador sobre qual o tipo de quantização que ele quer. Apenas é preciso fornecer o caminho para o ficheiro de áudio que se pretende quantizar e escolher o tipo de quantização.

```
# input pedido para selecionar método de quantização
Choose quantization option (0 for Mid-riser or 1 for Mid-tread):
```

Tendo o utilizador escolhido o tipo de quantização que pretende, são percorridas todas as samples do ficheiro e calculado um novo valor para elas. As fórmulas usadas para efectuar a quantização MidTread e MidRizer respectivamente foram[3]:

$$Q(x) = \Delta \cdot \left\lfloor \frac{x}{\Delta} + \frac{1}{2} \right\rfloor$$

$$Q(x) = \Delta \cdot \left(\left\lfloor \frac{x}{\Delta} \right\rfloor + \frac{1}{2} \right)$$

Após o cálculo do novo valor das samples, é definido o número de bits usados para codificar o ficheiro de áudio. Como os ficheiros de áudio fornecidos tinham todos uma *BitDeph* de 16 bits, e como a biblioteca *AudioFile* só suporta os valores 8, 16, 24 ou 32 foi tomada a decisão de reduzir para 8 bits.

Finalmente foi tomada a decisão de apresentar dois gráficos ao utilizador um com o sinal original e outro com o sinal quantizado para que seja mais fácil perceber o resultado final. Esses dois gráficos foram ambos desenhados com a biblioteca *matplotlib*.

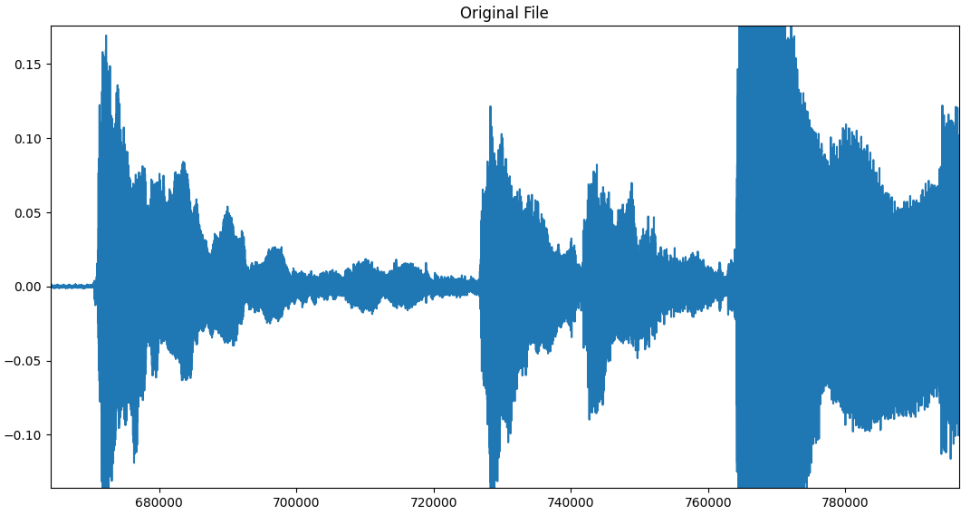


Figure 4.2: Excerto do sinal (sample02.wav) original

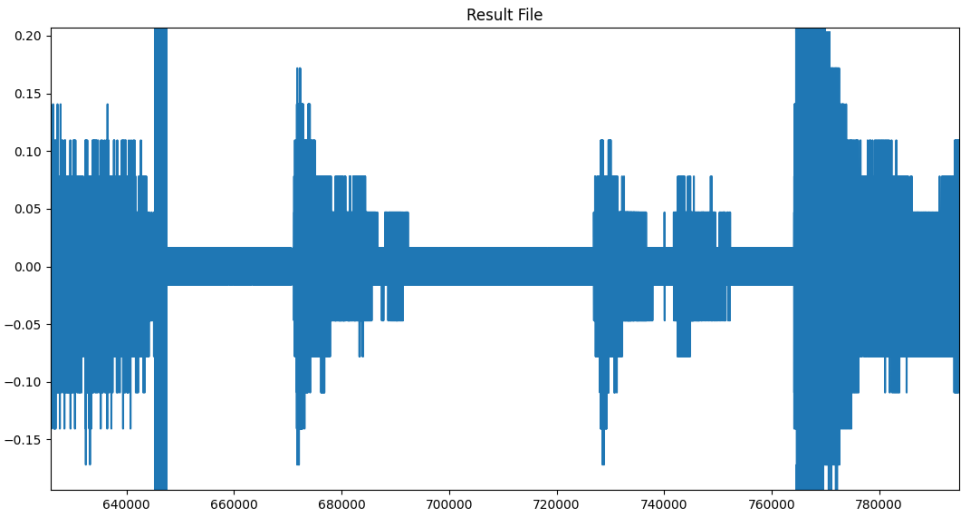


Figure 4.3: Excerto do sinal (sample02.wav) ao qual foi aplicado quantização Mid-Riser

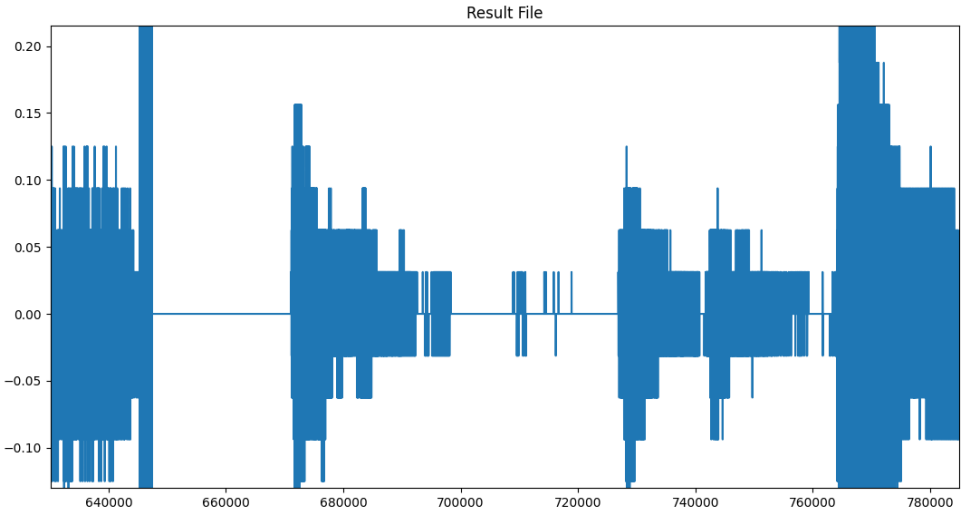


Figure 4.4: Excerto do sinal (sample02.wav) ao qual foi aplicado quantização Mid-Tread

4.2 Exercício 9

Este programa aceita como entrada o *path* para uma imagem e outro para a guardar a imagem transformada. Para a quantização foram usados os mesmo métodos que no exercício 8 (Secção 4.1).

Destacar que o bit depth da imagem transformada é fixo e tem valor 6.

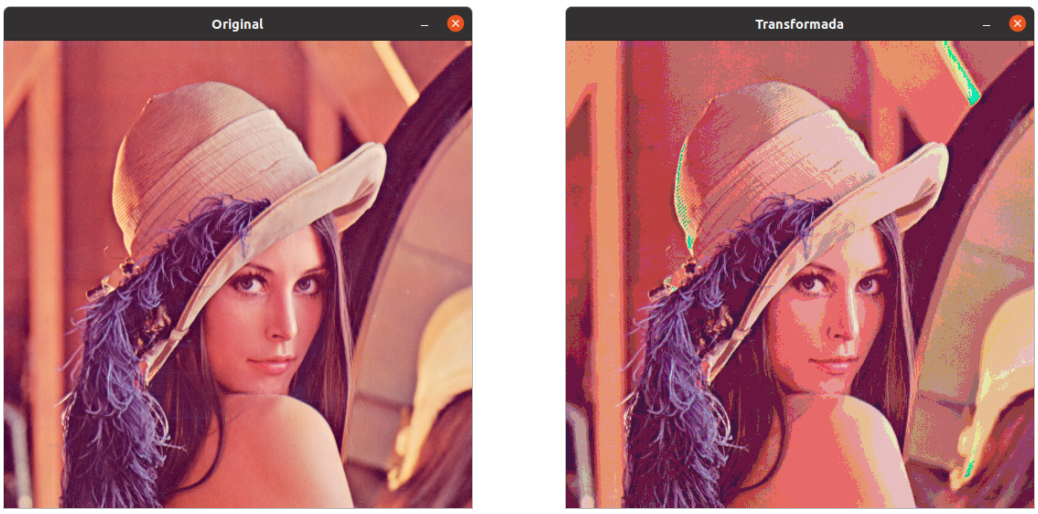


Figure 4.5: ./bin/ex9 files/miuda.png miuda_mid-riser.png

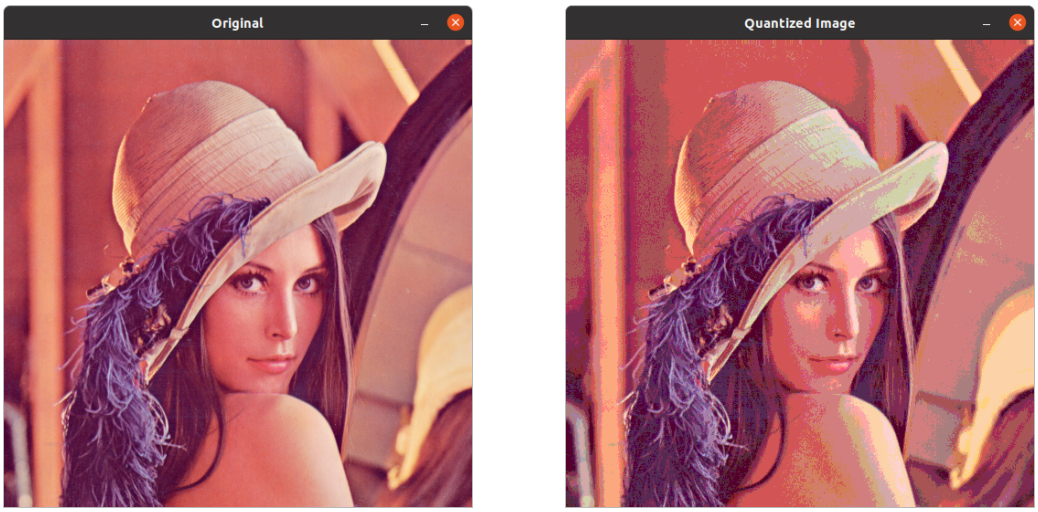


Figure 4.6: ./bin/ex9 files/miuda.png miuda_mid-tread.png

4.3 Exercício 10

[4] Como a operação de quantização introduz erros, torna-se importante medir a quantidade de ruído que foi introduzida. Para medir então a distorção (poder do ruído) introduzida usou-se a seguinte fórmula:

$$D = \frac{1}{N} \cdot \sum_{n=1}^{N} \left(x_n - \hat{x}_n\right)^2$$

Por sua vez para medir o poder do sinal sem distorção já se usou a fórmula:

$$\delta_x^2 = \frac{1}{N} \cdot \sum_{n=1}^{N} x_n^2$$

Finalmente para calcular o SNR (expresso em decibel) foi apenas necessário utilizar a fórmula

$$SNR = 10 \cdot \log_{10}\left(\frac{\delta_x^2}{D}\right)$$

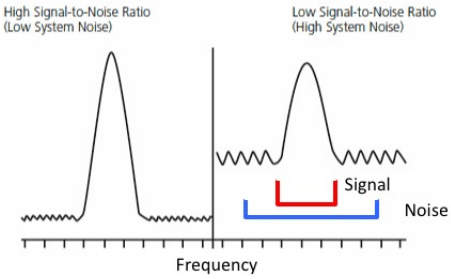


Figure 4.7: Referencia gráfica do impacto de ruído na onda eléctrica

Tendo estas fórmulas, para correr o programa e para as calcular é necessário que o utilizador forneça dois ficheiros (um "normal"e outro igual mas que foi comprimido). Ambos os ficheiro têm de existir e ter o mesmo número de samples. Se estas condições e verificarem, é feita uma iteração pelas samples dos ficheiros e calculado o SNR de acordo com as fórmulas anteriores.

Analisando a fórmula e significado do SNR é de esperar que quanto maior for o ruído de um ficheiro de áudio em relação ao original, menor será o SNR e quanto menor o ruído, maior será o SNR.

O máximo erro absoluto foi calculado enquanto estavam a ser percorridas todas as samples sequencialmente. Foi usada a fórmula:

$$e_a = \left| x_n - \hat{x}_n \right|$$

Adicionalmente a esta fórmula existia uma variável intitulada de *maxError* que armazenava o valor máximo do erro absoluto e que era actualizada sempre que necessário.

4.4 Exercício 11

Para o teste deste exercício foi usada uma imagem que foi comprimida com o programa do exercício 9 (Secção 4.2).

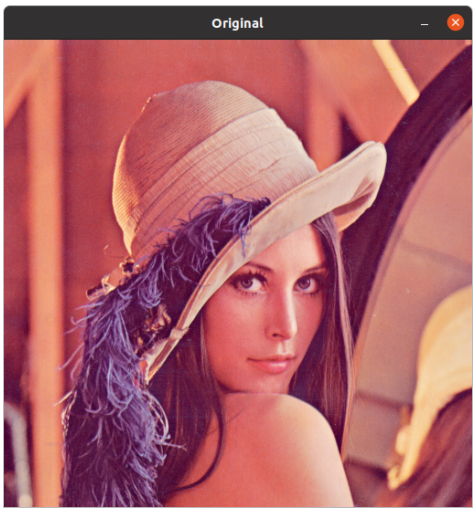
Este programa tem por base os métodos de cálculo do SNR do exercício anterior. Aceita como entrada o *path* para uma imagem original e outro para uma imagem que sofreu algum tipo de transformação/compressão. Como saída este programa apresenta na consola os valores calculados de SNR, PSNR e o máximo erro absoluto por pixel. O cálculo da distorção (que representa neste caso a média aritmética da distorção introduzida por cada canal de cor) foi realizado da seguinte forma:

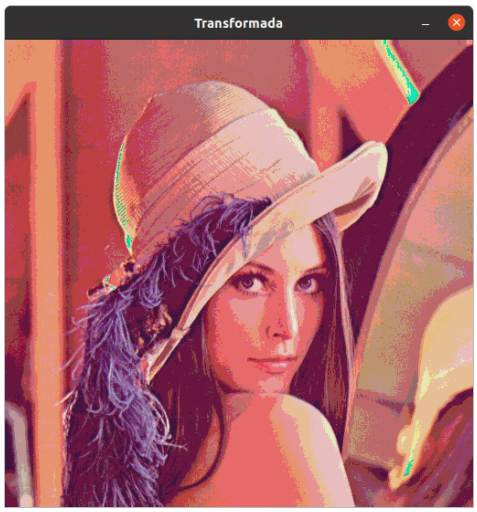
$$D = \sum_{n=1}^{ch} \cdot \left(\frac{1}{N} \cdot \sum_{n=1}^{N} \left(x_n - \hat{x}_n \right)^2 \right)$$

Foi também calculado o PSNR (dB) cuja fórmula é:

$$PSNR = 10 \cdot \log_{10}\left(\frac{x_{pp}^2}{D}\right)$$

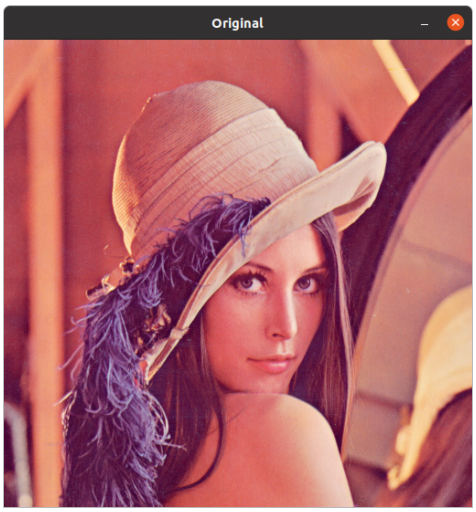
Esta grandeza difere do SNR no numerador da expressão. Neste caso é usado x_{pp} que representa a diferença entre picos opostos do sinal.

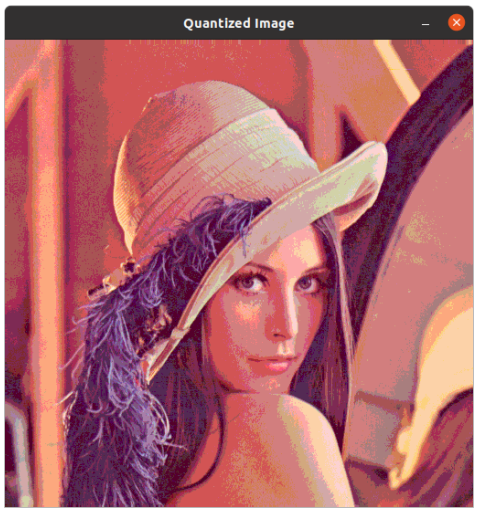




Transformada	**Valor**
SNR	75.905689 dB
PSNR	20.517824 dB
max. absolute error per pixel	238

Figure 4.8: ./bin/ex11 files/miuda.png miuda_mid-riser.png





Transformada	**Valor**
SNR	77.142100 dB
PSNR	23.170599 dB
max. absolute error per pixel	21

Figure 4.9: ./bin/ex11 files/miuda.png miuda_mid-tread.png

De ambas as figuras 4.8 e 4.9 podemos assumir que nesta imagem em concreto, o processo de quantização Mid-Rise induziu menos ruído na transformada por apresentar um valor menor do que a transformada da quatização Mid-Tread.

Ambos são usados para medir a qualidade da imagem resultante de uma transformação ou compressão. O SNR representa o quão forte é o sinal em relação à quantidade de ruído presente (porque o numerador é a média do erro ao quadrado), no entanto o PSNR representa o quão resistente é a imagem a perdas(devido ao ruído) em zonas de alta-intensidade de sinal(cor). [5]

Na análise da qualidade de imagem após a aplicação de um método de compressão chegamos à conclusão que é melhor usar a métrica PSNR por este método usar a amplitude máxima do sinal transformado ao invés da diferença para o sinal de entrada(SNR). Por outras palavras o PSNR, por ser normalizado pela amplitude máxima do sinal de origem torna-se mais independente da imagem original e, por isso, um método de comparação transversal entre várias imagens porque a imagem original não tem peso na definição da métrica. [6]

Capítulo 5

Contribuição dos autores

Todos os autores participaram de forma igual na divisão, desenvolvimento e discussão deste trabalho pelo que a percentagem de contribuição para cada aluno fica:

- Ana Filipe - 33.33 %
- João Gameiro - 33.33 %
- Pedro Abreu - 33.33 %

Bibliografia

[1] https://www.fourcc.org/fourcc.php.

[2] tutorialspoint. Digital communication - quantization. https://www.tutorialspoint.com/digital_communication/digital_communication_quantization.htm.

[3] Didattica in Rete. Laboratorio di algoritmi e strutture dati. https://www.dir.uniupo.it/pluginfile.php/154365/mod_resource/content/1/Lecture%204_6%20-%20Quantization%20and%20reconstruction.pdf.

[4] Informationskodning. Quantization. https://www.icg.isy.liu.se/courses/tsbk02/material/lect5-6.pdf.

[5] https://dsp.stackexchange.com/questions/11326/difference-between-snr-and-psnr.

[6] https://dsp.stackexchange.com/questions/26773/why-is-psnr-used-for-image-quality-metrics-i
noredirect=1&lq=1.