Universidade de Aveiro

Informação e Codificação
Projeto 2

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

Departamento de Electrónica, Telecomunicações e Informática

29 de dezembro de 2021

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

O presente relatório visa descrever a resolução do Projecto 2 desenvolvido no âmbito da unidade curricular de Informação e Codificação. O principal objetivo deste projeto era criar dois codificadores, um de imagem e outro de áudio com a possibilidade de a codificação ser efectuada de forma *lossless* ou *lossy*.

1.1 Compilação

O código desenvolvido para o projeto encontra-se disponível em: https://github.com/Torrakanor611/IC_project2.

```
#Estando no directório base do projeto
mkdir build && cd build

#Compilar o código
cmake ..
make

#Gerar e compilar documentação doxygen
doxygen doxyconfig
```

Os diretório do projeto incluem:

- diretório *src* -> como quatro pastas em que cada uma contém classes para implementar módulos do projeto (Bitstream, Golomb, codec de áudio e codec de imagem).
- diretório *test* -> com ficheiros cpp que implementam testes aos módulos desenvolvidos (testes aos codecs, teste ao golomb, bitstream, etc).
- diretório *files* -> com recursos que podem ser usados nos testes (imagens, ficheiros de áudio, etc).

Todos os ficheiros da pasta test, têm comentado no seu topo o comando que permite a sua compilação com as respetivas flags corretas.

Capítulo 2

Bitstream

Esta classe permite manipular um ficheiro dado bit a bit utilizando uma *stream*. O tipo primitivo mais pequeno possível representar C é o *char* e ocupa 1 byte em memória. Assim sendo, a menor unidade suportado em I/O é também 1 *byte*. No entanto, para acções de manipulação de ficheiros, neste caso compressão e descompressão, foi preciso criar esta classe de abstracção para I/O num ficheiro para manipular os ficheiros *bitwise*.

A assinatura pública desta classe é:

```
BitStream();
BitStream(const char* filename, char mode);
unsigned char readBit();
void writeBit(char bit);
void readNbits(char* bits, int n);
void writeNbits(char* bits, int n);
bool eof();
void close();
```

Um exemplo de uso desta classe é o ficheiro *copybinfile.cpp* na diretório *examples*.

Capítulo 3

Golomb

A codificação de *Golomb*, permite gerar um conjunto de códigos de tamanho variável livres de prefixo. A codificação envolve separar um inteiro em duas partes: a parte unária e a parte binária. Esta família de códigos apresenta resultados ideais para fontes de informação que são representadas por distribuições geométricas e depende essencialmente de um parâmetro $m > 0$.

[1] Considerando que se pretende codificar um número inteiro i, o processo de codificação segue os seguintes passos:

$$q = \left\lfloor \frac{i}{m} \right\rfloor$$
$$r = i - qm$$

O quociente q pode ter valores 0,1,2,... e vai ser codificado no seu respectivo código unário. Por sua vez o resto da divisão r pode assumir valores 0,1,2,...,m-1 e irá ser codificado pelo seu respectivo código binário.

Esta situação, descrita anteriormente, é válida para quando o valor de m é uma potência de 2 (pois se não o for o código binário não é eficiente). Se o m não for potência de 2 os passos para a codificação da parte binária são os seguintes:

- Definir $b = \lceil log_2 m \rceil$
- Se $r < 2^b - m$, codificar r em binário usando b-*1* bits.
- Caso contrário , codificar o valor $r + 2^b - m$ em binário com b bits.

Para o bom funcionamento desta classe, um dos atributos da mesma é um objecto *Bits-tream* no qual são escritas as codificações ou lidas para a sua consequente descodificação.

[2] Por sua vez a descodificação envolvia ler o valor escrito num ficheiro começando por contar o número de 1s (do código unário). Tendo o número de 1s (A) já conseguimos descobrir o tamanho do código que é A + b + 1 se m for uma potência de 2. Sendo assim neste caso os

restantes $c + 1$ bits representam R e convertendo esse número binário em decimal obtemos que o valor final do código é dado por mA + R.

Para valores de m que não são potências de 2, após contar o número de 1s, calculamos R em decimal que é representado pelos c-1 bits seguintes ao código unário. Finalmente se $R < 2^c - m$ o valor do código é mA + R caso contrário temos de considerar que R é os c bits seguintes ao código unário e o valor final descodificado é dado por mA + R $-(2^c - m)$.

Para podermos codificar valores negativos usando esta família de códigos, foi usado um método em que os números negativos são representados por números ímpares positivos e os números positivos são representados pelos pares. Esses métodos estão representados nas funções *fold* e *unfold* da classe *Golomb*.

Esta classe possui adicionalmente métodos que permitem codificar cabeçalhos que auxiliam na codificação e descodificação dos ficheiros nos codecs que vão ser descritos nas próximas secções (essas funções estão descritas na documentação doxygen).

Um exemplo do uso desta classe encontra-se no ficheiro *testGolomb.cpp*.

Capítulo 4

Codec de Áudio

4.1 Preditor

O preditor linear usado para o cálculo dos residuais é o que está descrito nas seguintes equações:

$$\begin{cases} \hat{X}_n^{(0)} = 0 \\ \hat{X}_n^{(1)} = x_{n-1} \\ \hat{X}_n^{(2)} = 2x_{n-1} - x_{n-2} \\ \hat{X}_n^{(3)} = 3x_{n-1} - 3x_{n-2} + x_{n-3} \end{cases}$$

Neste caso era apenas necessário especificar a ordem que se pretendia e de acordo com o valor dado era escolhido um destes preditores. Após o cálculo especificado no seguinte sistema de equações, faltava calcular o valor dos residuais que era dado por:

$$r_n = x_n - \hat{x}_n$$

Na reconstrução dos valores obtidos pelos preditores foram realizadas as operações inversas, que pressupunham que:

$$x_n = r_n + \hat{x}_n$$

Logo, era necessário calcular $\hat{x}_n$ que era dado pelos preditores especificado anteriormente mas desta vez de ser do valor das samples originais era dos valores obtidos nos residuais.

É importante referir que de modo a obter uma entropia mais baixa, foi tomada a decisão de fazer o processamento canal a canal, ou seja o preditor está logicamente dividido em dois, para poder fazer previsões só para o canal esquerdo, baseando-se apenas na informação do canal esquerdo e prever para o direito com base na informação do direito. No entanto, apesar da separação dos canais na previsão de valores, os residuais resultantes da previsão dos canais são colocados no mesmo vector.

4.2 Lossless Codec

Para testar este codec é necessário compilar e correr o ficheiro *test_ codecaud.cpp* com um argumento de entrada que é o caminho para o ficheiro a ser comprimido. Ao correr esse ficheiro vai ser apresentado um *prompt* ao utilizador para escolher o tipo de codec a usar. Basta escolher a opção correspondente a lossless. Para a implementação deste codec era necessário recolher os dados necessários para ser possível a reconstrução do ficheiro original no lado do descodificador. Logo, para esse efeito foram criadas funções adicionais na classe *Golomb* que permitem a codificação de cabeçalhos que vão conter informação crucial na reconstrução do ficheiro original. A informação contida num cabeçalho inclui:

```
Golomb m            -> 32 Bits
Numero de samples   -> 32 Bits
Sample Rate         -> 32 Bits
Format              -> 32 Bits
Channels            -> 4  Bits
```

Todos estes valores vão codificados em binário para que a sua posterior leitura e descodificação não esteja restringida à existência do m do *Golomb*.

Sendo assim, observando a Figura 4.1, conseguimos perceber melhor o codificador deste codec de áudio. O valor de cada sample é enviado para o predictor que por sua vez irá calcular um valor usado para obter os residuais r. Obtendo os residuais (r) falta apenas calcular o m ideal para codificar os residuais com códigos de *Golomb*. Foi tomada a decisão de que o cálculo do m ideal era dado por dois passos (cálculo da média aritmética dos valores dos residuais após a operação de *fold* descrita no capítulo Capítulo 3):

$$mean = \frac{\sum fold(r_n)}{N}$$

E finalmente obter o m com a expressão:

$$m = \left\lceil \frac{-1}{log_2\left(\frac{mean}{mean+1.0}\right)} \right\rceil$$

A decisão de usar a média aritmética teve por base a ideia de que sendo que os valores dos residuais são os que vão ser codificados após a operação de *fold*, o valor obtido para o m seria mais fiável que se fosse feita a média aritmética das samples do ficheiro de áudio.

Após ser calculado o m são criados os cabeçalhos e escritos no ficheiro, sendo que finalmente o único passo que falta é o de escrever os valores dos residuais codificados com códigos de *Golomb* no ficheiro destino. Essa ação é feita com sucessivas chamadas à função *encode* da classe *Golomb*.

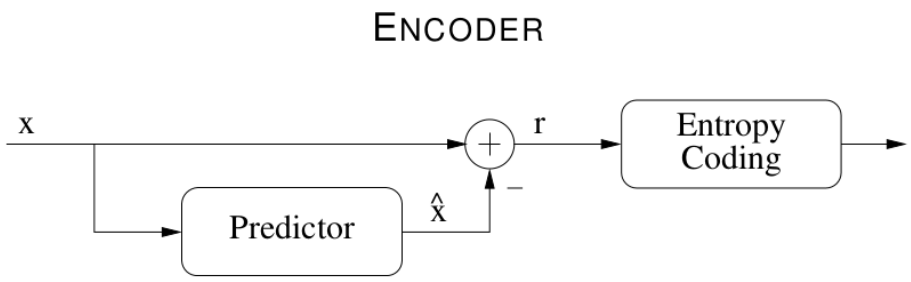

Figure 4.1: Estrutura lógica do Codificador

Por sua vez na figura 4.2 podemos observar o processo contrário ao que foi descrito anteriormente que é o de descodificação. Este processo é iniciado ao ler primeiro os cabeçalhos com as funções para tal efeito desenvolvidas na classe *Golomb*. Ao ler o cabeçalho obtemos não só o m mas também outros dados que nos permitem reconstruir o ficheiro original e saber quantos mais valores temos de ler. Tendo processado o número de samples, e o valor do m, precisamos agora de ler e descodificar todos os valores dos residuais calculados. Essa leitura é feita com sucessivas chamadas à função *decode* da classe *Golomb*.

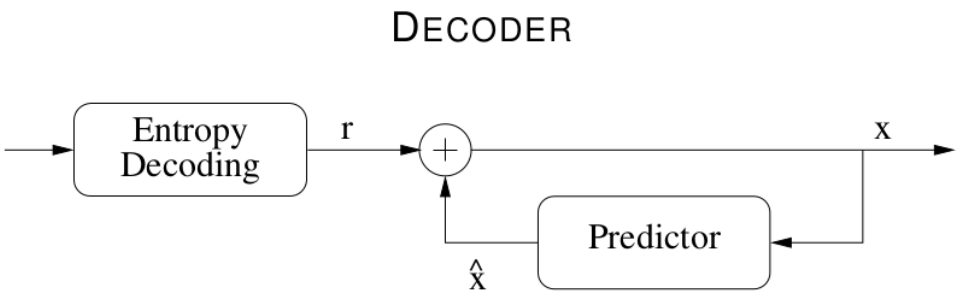

Figure 4.2: Estrutura lógica do *Decoder*

Após todo este processo estar concluido são copiados todos os dados obtidos no processamento de informação para um ficheiro àudio de output com recurso aos métodos da classe *libsndfile*.

4.3 Lossy Codec

Para testar este codec é necessário compilar e correr o ficheiro *test_ codecaud.cpp* com um argumento de entrada que é o caminho para o ficheiro a ser comprimido. Ao correr esse ficheiro vai ser apresentado um *prompt* ao utilizador para escolher o tipo de codec a usar. Basta escolher lossy e depois indicar o número de bits a ser remover.

No caso do codec lossy, a estrutura base é a mesma na medida em que são usados os mesmos preditores para codificação e descodificação de dados, assim como os mesmos métodos para obter o melhor valor de m para a codificação *Golomb*. Apenas foi feita uma alteração, que foi a adição de um processo de quantização aos residuais calculados com o preditor. No entanto foi também necessário alterar o preditor deste codec pelo facto de o cálculo dos residuais ter de refletir esta quantização que é realizada. Assim sendo sempre que era quantizado um residual, o valor usado nessa posição para o preditor é dado pelo soma de uma predição da sample com o seu residual quantizado.

A quantização é realizada com um shift à direita do número de bits indicado pelo utilizador. Quando se atualiza o valor da sample para fazer o sincronismo do erro (soma da predição com residual quantizado), é desfeito o shift (realizado na direcção oposta o mesmo número de vezes).

O processo de descodificação é precisamente igual ao que é realizado no caso do codec lossless com a exceção de que é necessário reconstruir a quantização que foi efectuada. Essa reconstrução é feita efectuando um shift na direção oposta o mesmo número de vezes que foi feito na codificação.

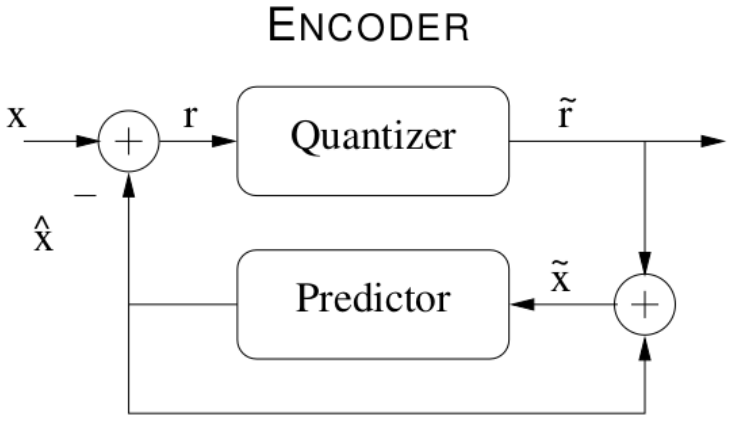

Figure 4.3: Estrutura lógica do Codificador do Codec Lossy

4.4 Resultados

Ao longo da próxima secção (e nos resultados apresentados no próximo capítulo) são apresentadas diversas vezes a taxa de compressão que foi obtida através da seguinte fórmula:

$$Compression\ Rate = \frac{UncompressedFile}{CompressedFile}$$

Na Tabela 4.1 é possível observar os resultados de vários testes efectuados para testar o codec de áudio lossless. Para além dos resultados dos testes é apresentado também o teste

de compressão para o codec de áudio FLAC para que pudéssemos comparar os resultados obtidos com alguma referência.

Ficheiro	**Preditor**	**Taxa de Compressão**	**Duração (s)**	**FLAC Compressão**	**FLAC Duração**
sample01.wav	1	1,358	1.2070	1,499	0,193
	2	1,409	1.1427		
	3	1,367	1.1979		
sample02.wav	1	1,385	0.5896	1,588	0,157
	2	1,377	0.5945		
	3	1,317	0.5947		
sample03.wav	1	1,466	0.7587	1,780	0.183
	2	1,526	0.7029		
	3	1,500	0.6975		

Tabela 4.1: Resultados do Codec Lossless

Em relação aos resultados das taxas de compressão foram bastante parecidos com os do FLAC, mas mesmo assim ficaram sempre abaixo do valor do codec. Comparando os resultados dos preditores, podemos concluir que aquele que teve a melhor taxa de compressão foi nos três casos o preditor 2.

Por sua vez a duração do processo de compressão já foi bastante superior ao tempo do FLAC em todos os casos. Em termos de comparação entre preditores os resultados são sempre bastante semelhantes não se destacando um dos outros por resultados melhores. Podemos também observar que o tempo que demorou a comprimir a sample01.wav foi maior que nos dois casos. Isto deve-se pelo facto de este ser maior que os outros ficheiros e consequentemente mais tempo demora a compressão. Tal facto também se refletiu nos tempos de compressãoo do FLAC.

Analisando agora a entropia antes e depois do processo de predição do codec lossless (no caso do preditor 2 pois este foi o que apresentou melhores resultados), podemos concluir que o preditor cumpriu com o seu objetivo. A principal finalidade de usar um preditor era obter uma sequência de valores cuja a entropia fosse menor que a sequência de valores original. [3] A entropia neste caso representava o número de símbolos binários necessários para codificar a fonte de informação. Logo sendo que o preditor reduziu a entropia, significa que são necessários menos bits para codificar os símbolos do ficheiro de áudio o que por sua implica que podemos atingir uma maior compressão.

Ficheiro	**Entropia Original**	**Entropia Residuais**
sample01.wav	13.9086	11.3135
sample02.wav	13.035	11.2006
sample03.wav	13.2678	10.3173

Tabela 4.2: Ficheiros de texto e a sua entropia

Podemos também observar que a sample03.wav foi a que registou uma entropia de residuais mais baixa. Esse facto é comprovado pela Tabela 4.1 em que se verifica que a sample03.wav foi dos 3 aquele que obteve uma maior taxa de compressão.

Comparando os resultados obtidos anteriormente com os do codec lossy (escolhendo o preditor 2 e indicando que o número de bits a remover é 2) podemos observar que a taxa de compressão foi maior que a do FLAC e do codec lossless em todos os ficheiros. A duração também foi ligeiramente menor que a do codec lossless assim como a entropia dos residuais. O facto de a entropia ser menor implica consequentemente uma maior taxa de compressão o que é comprovado pelos resultados obtidos.

Ficheiro	**Taxa de Compressão**	**Duração (s)**	**Entropia Residuais**
sample01.wav	1,7102	1.0039	9.31587
sample02.wav	1,6632	0.5671	9.21589
sample03.wav	1,8858	0.6843	8.33997

Tabela 4.3: Resultados do Codec Lossy escolhendo preditor 2 e removendo 2 bits

Na Tabela 4.4 está representado o SNR obtido ao comparar um ficheiro restaurado (após a compressão do codec lossy) com o seu original. Evidentemente podemos concluir que à medida que removemos mais bits mais pequeno é o SNR. Isto acontece pois ao remover bits estamos a introduzir erros na fonte de informação, o que compromete a sua qualidade e por sua vez aumenta o ruído e consequentemente diminui o SNR. Para os dois ultimos valores do número de bits removidos, o ficheiro descomprimido apresenta quase só ruído.

Número de Bits removidos	**SNR**
1	74.8448
2	66.3979
5	46.7108
6	40.5911
11	10.3909
12	4.35142
14	-8.22758
15	-14.813

Tabela 4.4: Resultados do Codec Lossy para vários valores de remoção de bits

Capítulo 5

Codec de Imagem

5.1 Preditor

O preditor usado para este codec foi não linear e as suas equações são:

$$\hat{x} = \begin{cases} min(a,b) & if \quad c \geq max(a,b) \\ max(a,b) & if \quad c \leq min(a,b) \\ a+b-c & otherwise \end{cases}$$

Este preditor tem por base a configuração espacial do JPEG:

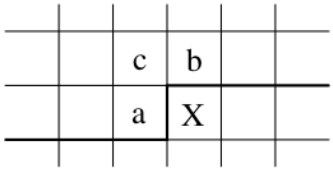

Figure 5.1: Configuração espacial do JPEG

5.2 Lossless Codec

A estrutura deste codec é baseada essencialmente na que foi descrita no capítulo Capítulo 4, sendo que a sua organização lógica é basicamente a mesma que a representada nas imagens Figura 4.1 e Figura 4.2. O preditor utilizado foi o descrito na secção anterior. A diferença neste codec para o áudio está no processamento.

A transformação rgb <-> YCbCr usada foi 8-bit full-range usada também no jpeg [4].

The possible range of values for chrominance and luminance reserves some footroom and headroom, which is necessary to provide some space for overshooting, e.g. in combination with analog video equipment. For computer based applications using RGB and YCbCr color formats, in many cases the complete possible range of 8 bit is used, without providing a footroom or headroom. Typically, this full-range color format is used for JPEG images.

The conversion of RGB colors into full-range YCbCr colors is described by the following equation:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Ranges:
R/G/B [0 ... 255]
Y/Cb/Cr [0 ... 255]

www.equasys.de

RGB to full-range YCbCr color conversion

The other way round, to convert a full-range YCbCr color into RGB is described by the following equation:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 0.000 & 1.400 \\ 1.000 & -0.343 & -0.711 \\ 1.000 & 1.765 & 0.000 \end{bmatrix} \cdot \begin{bmatrix} Y \\ (Cb - 128) \\ (Cr - 128) \end{bmatrix}$$

Ranges:
Y/Cb/Cr [0 ... 255]
R/G/B [0 ... 255]

www.equasys.de

Full-range YCbCr to RGB color conversion

Figure 5.2: Fórmulas usadas para conversão RGB <-> YCbCr

De seguida, foi necessário manipular o tamanho das componentes Cr e Cb, porque era necessário estas serem *sub-sampling* do tipo 4:2:0. O *resize* destas imagens foi feitas sem recurso a funções do OpenCV. As componentes Cb e Cr são constantemente 1 quarto do tamanho em área em relação à imagem principal, logo no *resize* RGB -> YCbCr é feita a média entre 4 valores adjacentes da mesma componente e é esse valor que representa essa componente no novo tamanho:

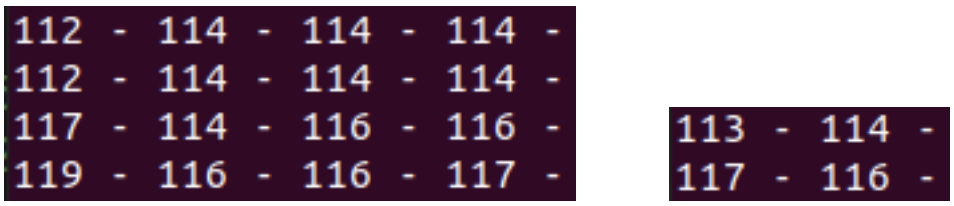

Figure 5.3: Exemplo do *resize* para 1/4 do tamanho, conversão RGB -> YCbCr (esq. -> dir.)

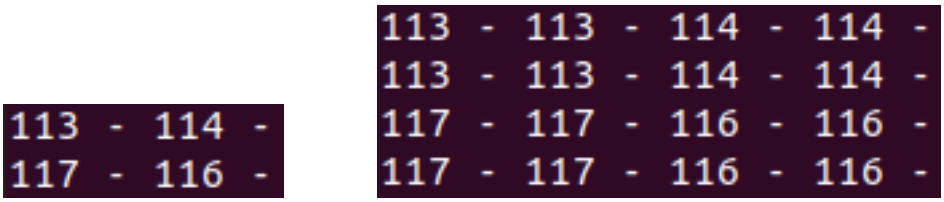

Figure 5.4: Exemplo do *resize* para 4* o tamanho, conversão YCbCr -> RGB (esq. -> dir.).

Além disso para visualizar e estudar a evolução do Codec de imagem, este dispõe de um

modo *verbose* passado no constructor da classe. Um objecto da classe inicializada em modo verbose aquando de uma compressão e descompressão de ficheiro irá apresentar em janelas auxiliares a imagem de entrada, o histograma dos valores RGB para a imagem original, o histograma do valores residuais e a imagem restaurada. Para além disto apresenta também o valor da entropia associada à imagem original(RGB) e dos valores residuais. A entropia foi calculada de acordo com a medida probabilística de informação de Shannon.

Para o cálculo do m do *Golomb* em vez de ter sido usada a média aritmética dos residuais após a operação de *fold* (como no caso do codec do áudio), foi usada a média ponderada dos residuais, pelo facto de terem sido obtidos melhores resultados com a mesma nesta caso. Todo o resto do processo de cálculo, após o cálculo da média foi igual ao áudio.

5.3 Lossy Codec

Tal como no caso anterior a estrutura base deste codec é a mesma que no caso do áudio, sendo que a única diferença era a informação de entrada e no modo de processar a mesma.

Ou seja, para desenvolver este codec, adicionou-se ao codec lossless o processo de quantização dos residuais que são produzidos a seguir ao processo de predição. Essa quantização consistiu basicamente em remover bits através da operação de shift à direita. O número de bits a remover é especificado pelo utilizador. Após o cálculo do residual quantizado é atualizado o pixel nessa posição para passar a ser o valor previsto somado com o seu residual quantizado. Nessa atualização o valor usado do residual quantizado é reposto com um shift igual ao anterior, mas na direcção oposta. No processo de descodificação do residual esse shift é desfeito tal como explicado anteriormente (feito na direcção oposta o mesmo número de vezes).

Todo o restante processo de descodificação é feito exatamente da mesma maneira que no codec lossless. É importante referir que o processo de quantização é exatamente o mesmo que é efectuado no codec de áudio.

5.4 Resultados

Para testar este codec é necessário compilar e correr o ficheiro *test_ codecimg.cpp* com um argumento de entrada que é o caminho para o ficheiro a ser comprimido. Ao correr esse ficheiro vão ser apresentados 2 *prompts* ao utilizador. O primeiro para perguntar se o utilizador quer usar o mode *verbose*, e o segundo para que o utilizador possa escolher o tipo de codec a usar. Se o utilizador escolher a opção lossy tem de indicar o número de bits a remover. (Nota: as janelas com gráficos *matplotlib* estão em modo blocking. Para continuar a execução do programa é preciso fechar estas janelas.) Para testar e validar o código desenvolvido são apresentados as seguintes resultados. Os dados são retirados do output do ficheiro anteriormente descrito que se encontra no directório *test*.

5.4.1 Lossless

Analisando os resultados obtidos observamos que o preditor produziu uma sequência de residuais com entropia bastante menor que a original, o que permite um taxa de compressão alta.

Ficheiro	Entropia	Entropia residuais	Taxa Compressão(%)	Duração (s)
miuda.png	7.621	4.280	2,1156	0.163
easter.png	7.417	3.191	1,7189	0,179

Tabela 5.1: Resultados do Codec Lossless

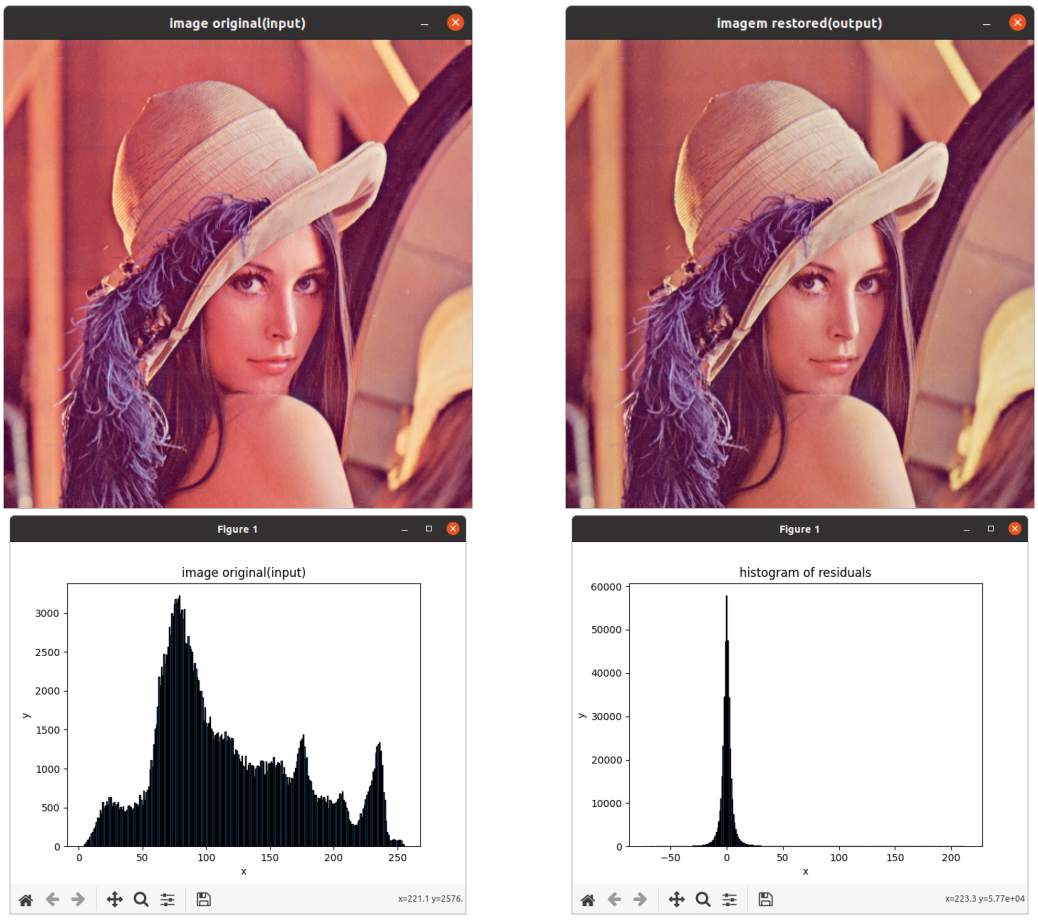

Figure 5.5: Resultados do ficheiro *test_ codecimg* aplicada à imagem miuda.png em modo *verbose*.

5.4.2 Lossy

De seguida são apresentados os resultados da compressão lossy aplicada à imagem miuda.png em modo *verbose*, para quando o número de bits a reduzir foi 3, 4 e 5.

Analisando os resultados da Tabela 5.2 e comparando com os valores obtidos para o codec lossless, verifica-se que a taxa de compressão é muito maior.

Nível de Quantização	**Entropia**	**Taxa de Compressão (%)**
3	1.716	5,0315
4	1.184	6,5183
5	0.721	7,8506

Tabela 5.2: Resultados do Codec Lossless

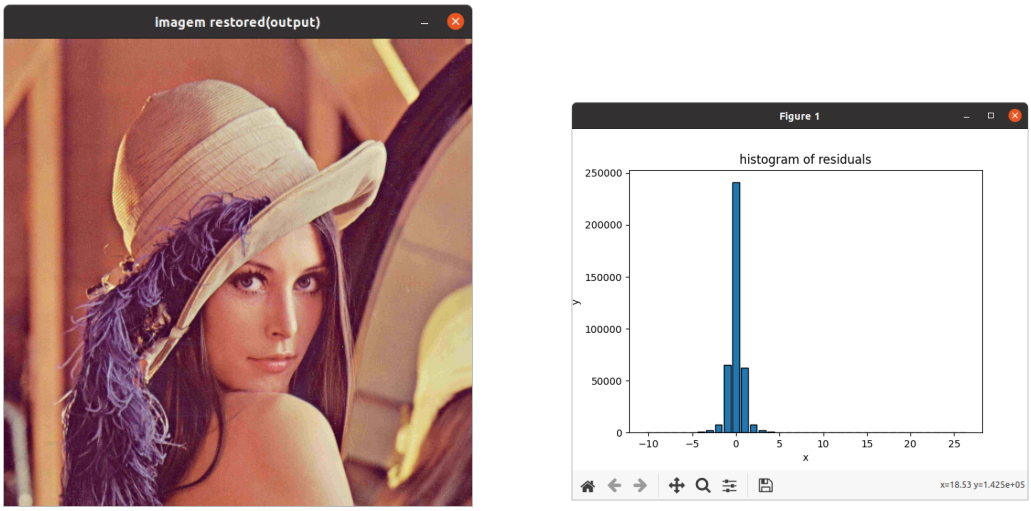

Figure 5.6: Resultados do ficheiro com 3 steps de quantização.

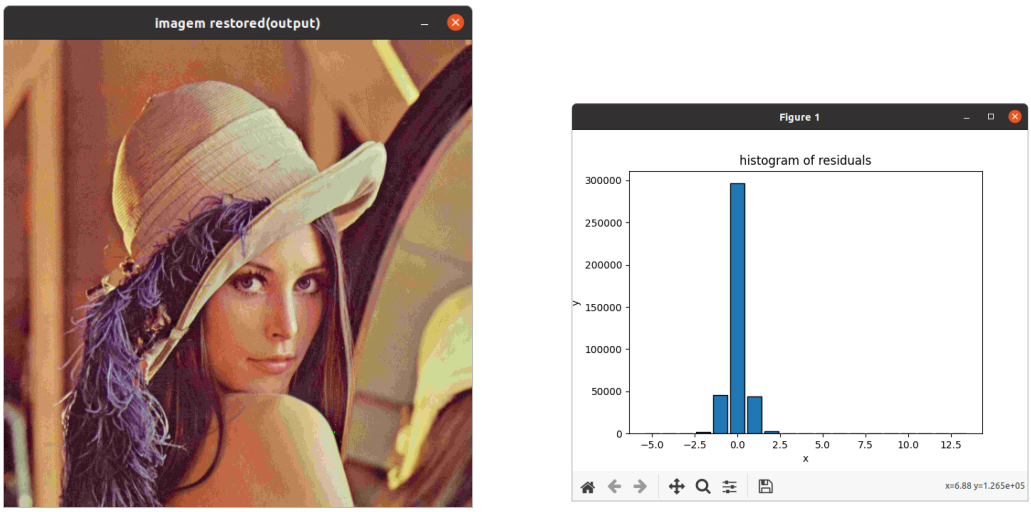

Figure 5.7: Resultados do ficheiro com 4 steps de quantização.

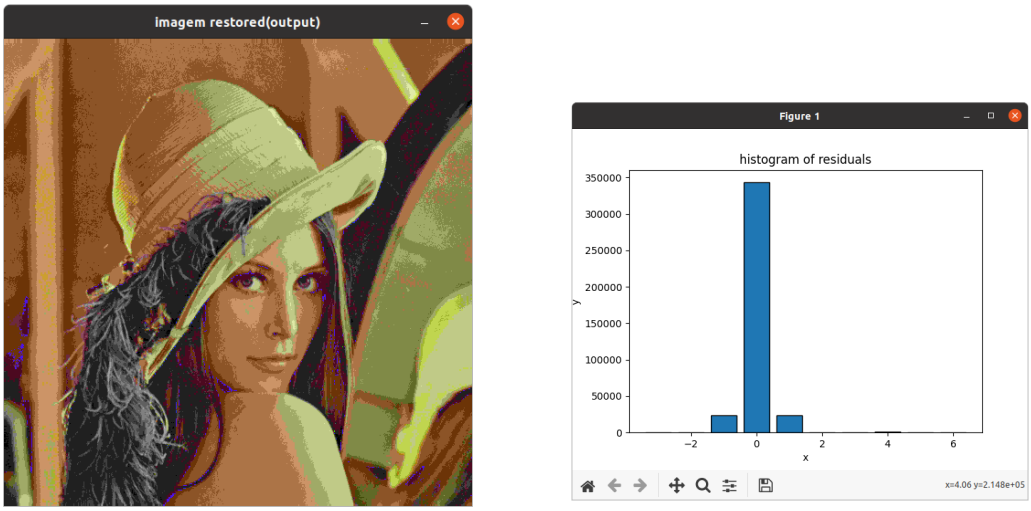

Figure 5.8: Resultados do ficheiro com 5 steps de quantização.

Podemos concluir que quanto maior o passo de quantização, maior a perda de informação e consequentemente menor qualidade na imagem. Como esperado a entropia é também menor quanto maior o nível de quantização.

Podemos observar também que quanto menor a entropia calculada nos residuais a serem codificados, maior é a taxa de compressão do ficheiro.

Capítulo 6

Contribuição dos autores

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

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

Bibliografia

[1] Armando J. Pinho. *Some Notes For the Course Information and Coding*, pages 62–63. Universidade de Aveiro, 2021.

[2] David Solomon. *Data Compression The Complete Reference, 3rd Ed*, pages 59–61. Springer, 2004.

[3] Khalid Sayood. *Introduction to Data Compression, 3rd Ed*, pages 16–17. Elsevier, 2006.

[4] `https://web.archive.org/web/20160127065924/http://www.equasys.de/colorconversion.html`.