Projeto 2 - IC

Universidade de Aveiro

João Torrinhas, Diogo Torrinhas, Tiago Bastos

Projeto 2 - IC

Departamento de Electrónica, Telecomunicações e Informática

Universidade de Aveiro

João Torrinhas, Diogo Torrinhas,
Tiago Bastos,
(98435) joao.torrinhas@ua.pt, (98440) diogotorrinhas@ua.pt
(97590) tiagovilar07@ua.pt

December 5, 2022

Contents

Chapter 1

Source Code

O código realizado pode ser encontrado em: https://github.com/diogotorrinhas/IC2

Chapter 2

Exercise 1

Neste exercício o objetivo é copiar uma imagem para um novo ficheiro.

Para tal, são passados como argumentos a imagem que se deseja copiar e o nome do novo ficheiro que irá conter a imagem copiada. Inicialmente é lida a imagem passada com argumento e por fim é copiada, pixel a pixel, para o novo ficheiro.

```
//Acessar aos pixeis da imagem e copiar para a nova
for(int i=0; i < image.rows; i++){
    for(int j=0 ; j < image.cols; j++)
        output.ptr<Vec3b>(i)[j] = Vec3b(image.ptr<Vec3b>(i)[j][0], image.ptr<Vec3b>(i)[j][1], image.ptr<Vec3b>(i)[j][2]);
}
```

Figure 2.1: copy image pixel by pixel

Chapter 3

Exercise 2

3.1 Point a)

Nesta alínea é pedido para criar a versão negativa de uma imagem.

O negativo da imagem é produzido subtraindo a cada pixel o valor máximo de intensidade. Por exemplo, para uma imagem de 8 bits, o valor máximo de intensidade é $2^8 - 1 = 255$.

Portanto, a cada pixel é subtraído 255 para produzir a imagem de saída negativa.

```
for(int i=0; i < image.rows; i++){
    for(int j=0 ; j < image.cols; j++)
        image.ptr<Vec3b>(i)[j] = Vec3b(255 - image.ptr<Vec3b>(i)[j][0], 255 - image.ptr<Vec3b>(i)[j][1], 255 - image.ptr<Vec3b>(i)[j][2]);
}
```

Figure 3.1: 255 - each pixel

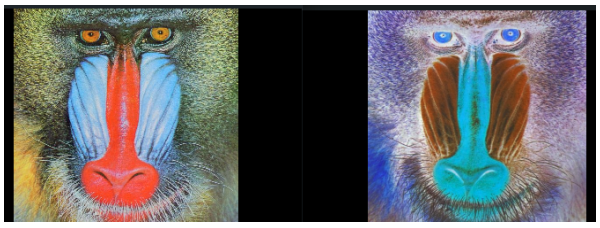

Figure 3.2: normal image and negative version

3.2 Point b)

Neste exercício o objetivo era rodar uma imagem na vertical ou na horizontal.

São passados como argumentos a imagem que se deseja rodar, o nome do ficheiro de saída, e o tipo de rotação(vertical ou horizontal). Para realizar a rotação foi usada a função *flip* da biblioteca *OpenCV*. Esta função roda uma imagem segundo o eixo do x ou o eixo do y.

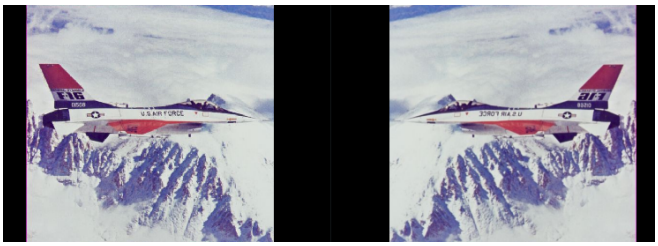

Figure 3.3: vertical mirror example

3.3 Point c)

Na alínea c) o objetivo é rodar uma imagem com angulos múltiplos de 90º(90, 180, 270).

Neste caso são passados como argumentos a imagem que se deseja rodar, o nome do ficheiro de saída, e o valor da rotação(90, 180, 270). Para realizar cada uma das diferentes rotações foi usada a função *flip*, referida anteriormente, e a função *transpose* da biblioteca *OpenCV*. Esta função gire uma imagem 90 graus no sentido anti-relógio. Como a função *transpose* roda uma imagem 90º no sentido oposto ao relógio, foi necessário usar a função *flip* para as rotações serem no sentido do relógio.

```
if(tipo == "90"){
    transpose(image, image);        //rotate 90
    flip(image, image, +1);
}
if(tipo == "180"){                  //rotate 180
    flip(image, image, -1);
}

if(tipo == "270"){                  //rotate 270
    transpose(image, image);
    flip(image, image, 0);
}
```

Figure 3.4: code to rotate for each angle

Figure 3.5: example of a 270^{o} image rotation

3.4 Point d)

Na última alínea o objetivo é aumentar ou diminuir a intensidade da luz de uma imagem.

São passados como argumentos a imagem que se deseja alterar a intensidade, o nome do ficheiro de saída, e o valor da intensidade desejada. Pode ser negativo para reduzir a intensidade, por exemplo -70, ou positivo para aumentar, 70. Foi usada a função *convertTo* da biblioteca *OpenCV* para alterar a intensidade da imagem.

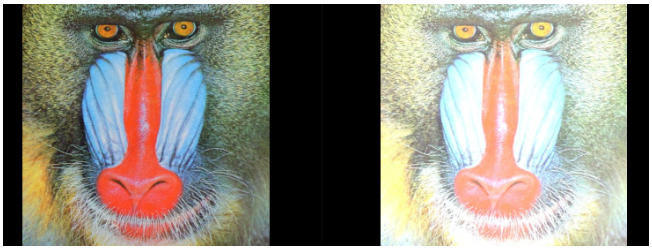

Figure 3.6: change intensity to value 100

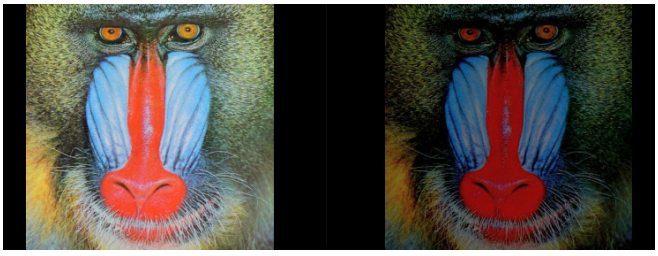

Figure 3.7: change intensity to value -100

Chapter 4

Exercise 3

A codificação de Golomb permite gerar um conjunto de códigos de tamanho variável livres de prefixo. Esta 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 e um parâmetro n.

Para ser possível codificar valores negativos, foi usado dois métodos, fold() e unfold(). Os números negativos são representados por números ímpares(positivos) e os números positivos são representados por números pares(positivos).

Se o valor 'i' for positivo, faz o fold() multiplicando esse valor por 2 e faz a sua respetiva codificação. Se o valor 'i' for negativo, faz o fold() multiplicando esse valor(absoluto) por 2 e subtraindo 1 e faz a sua respetiva codificação. Depois na descodificação, é usado o método unfold(), que dependendo se o valor descodificado é par ou ímpar, são feitas as contas inversas ás que foram feitas no fold(), para obter o valor descodificado correto.

Considerando que se pretende codificar um número inteiro i, o processo de codificação segue o seguinte fluxo:

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

$$r = i - qm$$

O quociente 'q' pode ter valores 0,1,2,3,... e vai ser codificado em código unário (p.ex. q=5, logo -> 000001). O resto da divisão 'r' pode assumir valores 0,1,2,3,...,m-1 e irá ser codificado em código binário (p.ex. r=2, logo -> 10).

Esta situação, é válida para quando o valor de m é uma potência de 2 (se não o for, o código binário não é eficiente). Se o m não for potência de 2 o processo de codificação da parte binária segue o seguinte fluxo:

- Definir b = [log2m]

- Se r $< 2^b$ - m
- Caso contrário, codificar o valor r + 2^b - m em binário com b bits

Nesta classe desenvolvida, um dos atributos da mesma é um objeto da classe BitStream no qual são escritas as codificações ou são lidas para a respetiva descodificação

A descodificação envolve ler o valor escrito num ficheiro começando por contar o número de 0s (da parte unária) até ao aparecimento do primeiro 1 (indicando o fim da parte unária). Tendo o número de 0s (A) já se consegue 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 obtem-se 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 0s até ao aparecimento do primeiro 1, 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 tem de se considerar que R é os c bits seguintes ao código unário e o valor final descodificado é dado por mA + R (2^c m).

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

Os comandos de compilação são os seguintes:

- g++ BitStream.cpp GolombTest.cpp -o testGolomb
- ./testGolomb

```
osboxes@osboxes:~/Desktop/IC_PROJETO2$ ./testGolomb
Insert m: 4
Insert n: 19

Encoding a ser feito...
Value to be encoded before Folding: 19
Value to be encoded after Folding: 38
Encoded Value: 000000000110

Decoding a ser feito...
Decoded Value: 19
osboxes@osboxes:~/Desktop/IC_PROJETO2$
```

Figure 4.1: example for m=4 , n=19

```
osboxes@osboxes:~/Desktop/IC_PROJETO2$ ./testGolomb
Insert m: 4
Insert n: -25

Encoding a ser feito...
Value to be encoded before Folding: -25
Value to be encoded after Folding: 49
Encoded Value: 000000000000101

Decoding a ser feito...
Decoded Value: -25
osboxes@osboxes:~/Desktop/IC_PROJETO2$
```

Figure 4.2: example for negative number; m=4 , n=-25

Chapter 5

Exercise 4

5.1 Preditor

O preditor, que foi usado para o cálculo dos residuais, é o que está descrito no seguinte sistema de equações:

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

Como há mais do que uma equação, é apenas necessário escolher qual destas equações se vai usar e de acordo com um valor introduzido é escolhido uma delas. Após o cálculo, falta apenas calcular o valor dos residuais, que é descrito pela seguinte fórmula:

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

Na reconstrução dos valores obtidos pelos preditores, usámos a seguinte equação:

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

Logo, é necessário calcular o valor de x̂n que é dado pelos preditores especificado anteriormente, mas em vez de usarmos o valor das samples originais, usa-se o valor obtido nos residuais.

De maneira a fazermos "inter-channel prediction" separámos o preditor logicamente em dois para podermos fazer previsões para o canal SIDE(L-R) e o canal MID(L+R/2). No entanto, apesas de usarmos dois canais na previsão de valores, os residuais resultantes, da previsão de canais, são colocados no mesmo vetor.

5.2 Lossless Codec

Para a implementação deste codec era necessário recolher a informação necessária para ser possível a reconstrução do ficheiro original no lado do descodificador. Logo, foram criadas várias funções na class Golomb que permitem a codificação de cabeçalhos que vão conter a informação essencial na reconstrução do ficheiro original.

De maneira à leitura e descodificação não estar restringida à existência do m no Golomb, todos estes valores são codificados em binário.

Posto isto, após o valor de cada sample ser enviado para o preditor, que por sua vez vai calcular um valor que vai ser usado no cálculo dos residuais r, falta apenas calcular o m ideal. Logo, para o cálculo do mesmo, seguimos as seguintes expressões:

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

Trata-se da média aritmética (Link para a wikipédia) dos valores dos residuais. Usámos a operação fold porque no golomb apenas passam inteiros positivos.

Em seguinda, é mostrado como obtemos a expressão final para o cálculo do m:

$$mean = \frac{(1-p)}{p} \tag{5.1}$$

$$\alpha = 1 - p \tag{5.2}$$

$$p = 1 - \alpha \tag{5.3}$$

$$m = \left\lceil -\frac{1}{\log \alpha} \right\rceil$$

Figure 5.1: initial formula for m

Deduzindo (5.1) e (5.2) e (5.3) obtemos as seguintes expressões:

$$mean = \frac{\alpha}{1-\alpha} \tag{5.4}$$

$$\alpha = \frac{mean}{mean+1} \tag{5.5}$$

E finalmente, substituindo (5.5) na fórmula inicial do m, obtemos a expressão final para o m:

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

Figure 5.2: final formula for m

Após ser calculado o m, são criados os cabeçalhos e escritos no ficheiro. Posto isto, falta apenas escrever os valores dos residuais codificados com códigos de Golomb no ficheiro destino, através de sucessivas chamadas à função *encode* da class Golomb.

No processo de descodificação lê-mos primeiro os cabeçalhos e obtemos não só o valor de m como também outros valores que nos permitem reconstruir o ficheiro inicial. Tendo obtido o número de samples e o valor de m, falta apenas saber os valores dos residuais e para isso temos de ler e descodificar os valores dos residuais, que tinham sido calculados na compressão, através de sucessivas chamadas da função *decode* da classe Golomb e, por fim, na reconstrução dos valores obtidos usamos a fórmula do capítulo 5, secção 5.1.

Após o processo anterior ter sido efetuado são copiados todos os dados obtidos desse processo para um ficheiro de áudio de output através de vários métodos da classe *libsndfile*.

5.3 Tests

Para testar este codec é necessário compilar o ficheiro *test_ audioCodec.cpp* e correr o executável resultante da compilação com um argumento de entrada que é o ficheiro original a ser comprimido, no caso do exemplo é *samples.wav*. Ao correr o ficheiro vai ser apresentado no prompt o tipo de codec que o utilizador quer correr. Basta escolher a opção de lossless e após o ficheiro ter terminado de correr vai ser criado um ficheiro *out.wav* onde é possível verificar que o áudio de saída é exatamente igual ao áudio original, como era suposto. É possível verificar também que após a compressão, o ficheiro resultante da mesma, *compress.bin*, é mais pequeno que o original o que mostra que está de acordo com o esperado.

```
joao@joaoT-pc:~/Desktop/Mestrado/IC/ProjetoZ$ g++ BitStream.cpp test_audioCodec.cpp -o test_audioCodec -lsndfile
joao@joaoT-pc:~/Desktop/Mestrado/IC/ProjetoZ$
```

Figure 5.3: compilation process

```
joao@joaoT-pc:~/Desktop/Mestrado/IC/ProjetoZ$ ./test_audioCodec sample.wav
Choose codec option (0 for lossless or 1 for lossy): 0
Choose the predictor (1, 2 or 3): 2
Start encoding...
... done encoding

start decoding...
... done decompress

joao@joaoT-pc:~/Desktop/Mestrado/IC/ProjetoZ$
```

Figure 5.4: run the executable resulting from the compilation

Chapter 6

Exercise 5

6.1 Preditor

Para este codec a estrutura do preditor é a mesma do lossless assim como os mesmos métodos para obter o melhor valor de m para a codificação de Golomb. Foi feita apenas uma alteração, que foi a adição de um processo de quantização aos residuais calculados com o preditor.

6.2 Lossy Codec

A quantização é feita com um shift de n bits à escolha do utilizador. O shift vai ser desfeito quando se atualiza o valor da sample (predição + residual quantizado).

O processo de descodificação é precisamente igual ao do lossless com a exceção de agora ter de se reconstruir também a quantização que foi efetuada através de um shift na direção oposta o mesmo número de vezes que foi feito na codificação.

6.3 Tests

Para testar este codec é fazer o mesmo que foi feito para testar o codec lossless à exceção q é preciso escolher a opção de lossy em vez de lossless e escolher também o número de bits que queremos remover, tal como mostra na figura 6.1. É possível verificar que o ficheiro de saída, resultante do comando expresso na figura 5.4, tem um pouco de ruído e quantos mais bits se quantizar mais ruído irá ter, ou seja, está de acordo com o que era previsto para compressão lossy. Verificámos também que o ficheiro resultante da compressão, *compress.bin*, é menor que o ficheiro original e é menor que o ficheiro comprimido usando compressão lossless, o que mostra que também está de acordo com o previsto.

```
joao@joaoT-pc:~/Desktop/Mestrado/IC/Projeto2$ ./test_audioCodec sample.wav
Choose codec option (0 for lossless or 1 for lossy): 1
Choose the predictor (1, 2 or 3): 2
Choose quantization step (number of bits to remove): 12
Start encoding...
... done encoding

start decoding...
... done decompress

joao@joaoT-pc:~/Desktop/Mestrado/IC/Projeto2$
```

Figure 6.1: Example for choosing the bits to remove

Chapter 7

Exercise 6

7.1 Preditor

Para o lossless image codec foi usado um preditor não linear em que as suas equações são:

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

Figure 7.1: Preditor

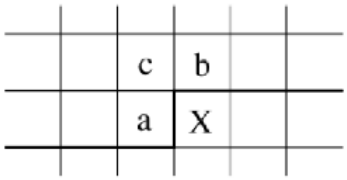

Figure 7.2: JPEG

Neste preditor baseámo-nos na ideia de que há 4 situações possíveis para obter os valores de a, b e c. Caso o x esteja no canto superior esquerdo a = 0, b = 0 e c = 0, caso x esteja na linha de cima , sem contar com o canto superior esquerdo, a = valor do píxel, b = 0 e c = 0, caso x esteja na coluna mais à esquerda, sem contar novamente com o canto superior esquerdo, a = 0, b = valor do píxel, c =

0 e finalmente caso não seja nenhuma das condições acima descritas, a = valor do píxel, b = valor do píxel, c = valor do píxel.

7.2 Lossless Image Codec

A estrutura deste codec foi baseada na estrutura do codec de áudio realizado anteriormente, capítulo 5, e o preditor usado é o que está descrito na secção acima.

A ideia inicial foi separar a imagem lida como argumento em 3 canais RGB, canal R, canal G, canal B. Em seguida, para cada um deles, foi feito o *encode* obtendo a imagem comprimida. Por fim, tendo a imagem comprimida, foi feito o *decode* de cada um dos canais, usando o inverso do preditor e os residuais que foram obtidos após a descodificação, obtendo assim a imagem final descomprimida.

Para o cálculo do m de *Golomb* foi usada, tal como anteriormente no codec de áudio, a média aritmética dos residuais após a operação de *fold*.

7.3 Tests

Para testar este codec de imagem é necessário compilar o ficheiro *test_ Codecim.cpp* e correr o executável resultante da compilação com um argumento de entrada que é a imagem a ser comprimida e depois descomprimida. Em seguida basta comparar ambas onde se pode ver que estão iguais, como era previsto.

```
diogo@DiogoT-PC:~/Desktop/project2$ g++ BitStream.cpp test_Codecim.cpp -o test_C
odecim $(pkg-config --libs --cflags opencv4)
diogo@DiogoT-PC:~/Desktop/project2$ ./test_Codecim image1.jpeg
... compress done ...

... decompress done ...
```

Figure 7.3: compilation process

Figure 7.4: input image and uncompressed image

Chapter 8

Authors' Contribution

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

- João Torrinhas - 33,33%
- Diogo Torrinhas - 33,33%
- Tiago Bastos - 33,33%

Bibliography

[1] Armando J. Pinho, Some Notes For the Course Information and Coding Universidade de Aveiro, 2022.

[2] https://en.wikipedia.org/wiki/Geometric_distribution

[3] https://answers.opencv.org/question/59529/how-do-i-separate-the-channels-of-an-rgb-image-and-save-each-one-using-the-249-version-of-opencv/