Informação e Codificação

Projeto 2

Universidade de Aveiro

Bruno Silva (97931)brunosilva16@ua.pt
Marta Oliveira (97613) marta.alex@ua.pt
Mariana Silva (98392) marianabarbara@ua.pt

VERSAO 1

Informação e Codificação

DETI

Universidade de Aveiro

Bruno Silva (97931)brunosilva16@ua.pt
Marta Oliveira (97613) marta.alex@ua.pt
Mariana Silva (98392) marianabarbara@ua.pt

04 de dezembro de 2022

Índice

Capítulo 1

Introdução

O presente relatório tem como objetivo descrever a resolução do Projeto 2 desenvolvido no âmbito da unidade curricular de Informação e Codificação.

O código desenvolvido para o projeto encontra-se disponível em `https://github.com/brunosilva16/IC/tree/main/proj2`

Para este projeto utilizamos a biblioteca *openCV*.

No ficheiro README.md no repositório estão as indicações de como compilar cada exercício.

Capítulo 2

Parte I

2.1 Exercício 1

No exercício 1 era pedido que se implementasse um programa que fizesse a cópia de uma imagem, píxel por píxel, de um ficheiro para o outro. Para isso, necessitamos de um ficheiro de imagem (existente) para entrada e um nome de um ficheiro para ser criado como *output*. Dessa forma, utilizámos da biblioteca *OpenCV*.

Uma imagem é a representação de um conjunto de pontos definidos por valores numéricos, formando uma matriz onde cada ponto é um píxel.

Cada ponto de uma imagem é decomposto em uma tripla de cores, isto é, representam-se usando o modelo RGB(Red, Green, Blue).

Para conseguirmos representar as imagens criámos 2 objetos do tipo *Mat*. Um representa a imagem original e outro vai representar a cópia (output).

Com o auxílio do tipo de dados *Vec3b*, isto é, um vetor com 3 bytes de entrada onde cada byte vai representar a intensidade de cada canal de cor(vermelho, verde e azul), fornecendo um alcance de 256 possíveis valores, ou intensidades, para cada tom, conseguimos copiar píxel a píxel de uma imagem paraa outra.

Com isto, realizámos o seguinte:

```
1  for(int i=0; i < image.rows; i++){
2        for(int j=0 ; j < image.cols; j++)
3            output.ptr<Vec3b>(i)[j] = Vec3b(image.ptr<Vec3b
                 >(i)[j][0], image.ptr<Vec3b>(i)[j][1], image.
                 ptr<Vec3b>(i)[j][2]);
4    }
```

Listing 2.1: Código fonte em C++

Através dos ciclos *for* vão ser percorridos todos os píxeis da nossa imagem.

De seguida, a informação obtida em cada uma das iterações dos ciclos é guardada num novo ficheiro de imagem.

Como compilar: make copy_image

Executar o programa: ../opencv-example-bin/copy_image <input_file> <output_file>

2.2 Exercício 2

Para este exercício implementámos um programa, igualmente utilizando a biblioteca *OpenCV*, que produz vários efeitos.

1. **Criar uma versão negativa de uma imagem.**

   Para criar uma versão negativa de uma imagem temos que subtrair, a cada pixel que a constitui, o valor máximo de intensidade que é possível. Neste caso, este valor é 255, pois como explicámos no exercício anterior existe um alcance de 256 valores possíveis para cada tom.

```
1     if(effect=="n"){
2         Mat img2=255 - image; //Image negative is
              produced by subtracting each pixel from the
              maximum intensity value. e.g. for an 8-bit
              image, the max intensity value is 28    1 =
              255,
3         //thus each pixel is subtracted from 255 to
              produce the output image
4         imshow("img2",img2);
5         waitKey(0);
6         return 0;
7     }
```

Listing 2.2: Código fonte em C++ para criar o negativo da imagem

Compilar programa: make ex_2

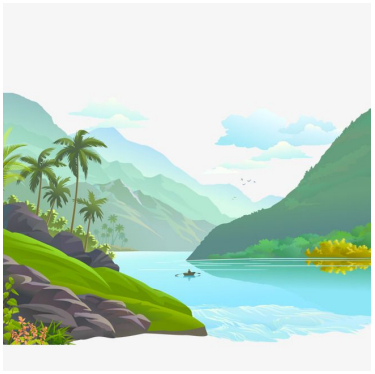
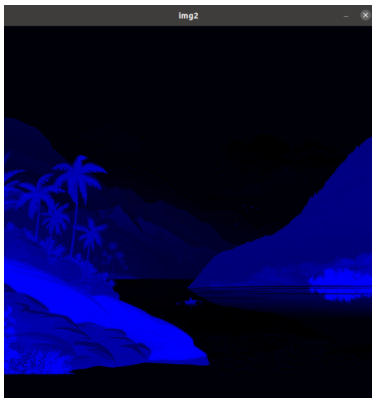


Imagem Original



Negativo da Imagem

Executar programa: ../opencv-example-bin/ex_2 <input_file> n

2. **Criar uma versão espelhada de uma imagem (horizontalmente ou verticalmente).**

Para criar uma imagem espelhada, usufruimos de uma função própria do *OpenCV*: *flip*.

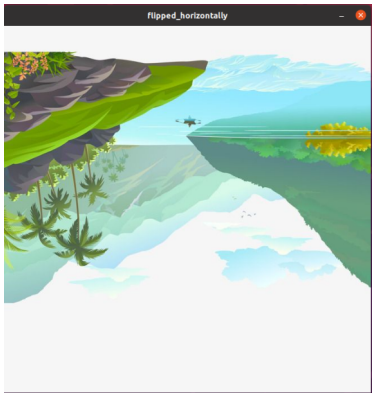
Esta função usa um *flip code* para espelhar a imagem. Para espelhar no eixo do x coloca-se um '0' e no eixo dos y coloca-se um '1'.

Usando a mesma imagem para efeitos de testes , de seguida apresentamos os resultados da execução do código para ambos os tipos de espelhagem (horizontal e vertical).

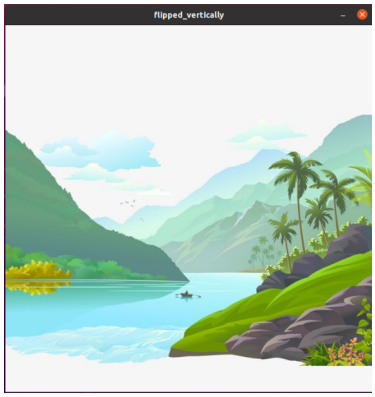
```
1     if(effect=="m"){ //https://stackoverflow.com/
          questions/14907964/mirror-image-in-opencv
2         Mat dst2;
3         cout<<"To flip image horizontally insert a '0'
              to flip vertically insert an '1'\n";
4         int value;
5         cout<<"choose value: "<<endl;
6         cin>>value;
7         //flip code: A flag to specify how to flip the
              array; 0 means flipping around the x-axis
              and positive value (for example, 1) means
              flipping around y-axis.
8         // //Negative value (for example, -1) means
              flipping around both axes. Return Value: It
              returns an image.
```

```
9           if(value==0){
10              Mat dst2;
11              flip(image,dst2,0);
12              imshow("flipped_horizontally",dst2);
13              waitKey(0);
14          }
15          if(value==1){
16              Mat dst2;
17              flip(image,dst,1);
18              imshow("flipped_vertically",dst);
19              waitKey(0);
20          }
21      }
```

Listing 2.3: Código fonte em C++ para espelhar a imagem



Flipped Horizontally



Flipped Vertically

```
2/opencv-example-src$ ../opencv-example-bin/ex_2 teste m
To flip image horizontally insert a '0'. To flip vertically insert an '1'
choose value:
1
```

Exemplo de compilação para criar versão espelhada

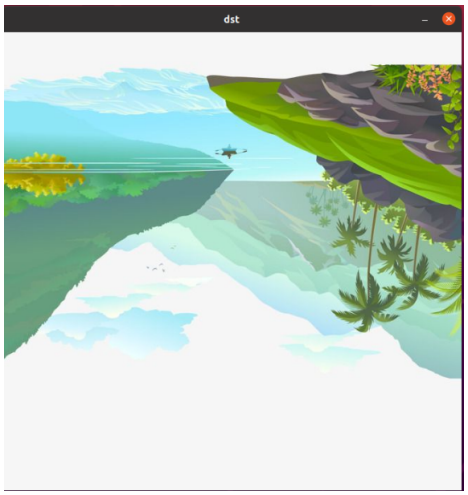
3. **Rotacionar uma imagem num grau múltiplo de 90.**

Neste efeito, mais uma vez, usufruimos de uma função própria do *OpenCV*: *rotate*.

```
1       if(effect=="r"){
2
```

```
3           if(r=="90"){
4               rotate(image,dst,ROTATE_90_CLOCKWISE);
5           }
6           else if(r=="-90"){
7               rotate(image,dst,ROTATE_90_COUNTERCLOCKWISE)
                    ;
8           }
9           else if(r=="180"){
10              rotate(image,dst,ROTATE_180);
11          }
12
13      imshow("dst", dst);  //displaying output image file
14      waitKey(0);          //to exit press escape
15
16      }
```

Listing 2.4: Código fonte em C++ para rodar a imagem



Rotação a 180º graus

Executar programa: ../opencv-example-bin/ex_2 <input_file> r

```
marta@marta-TUF-Gaming-FX505GD-FX505GD:~/Desktop/Universidade/IC/proj2/opencv-example-src$ ../opencv-example-bin/ex_2 test r
choose angle of rotation: 90,-90 or 180
180
```

Exemplo de compilação para criar versão rotacionada

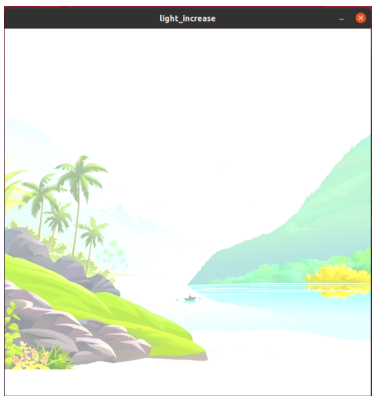
4. **Aumentar ou diminuir intensidade de luz da imagem.**

Nesta implementação, usamos a função *convertTo* do *OpenCV*. Neste caso, os parâmetros de entrada da função são: o ficheiro de imagem destino onde irá ser guardado o resultado do efeito, -1 dado que o número de canais da matriz de imagem destino vai ser igual à origem, 1 pois não queremos alterações na escala, e *value* que pode ser um valor negativo ou positivo.

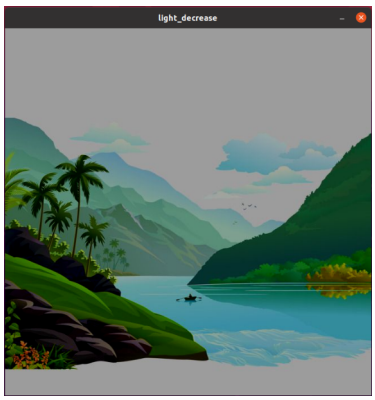
Este *value* vai ser adicionado a cada píxel. Se este for negativo vai haver uma diminuição de brilho, caso contrário vai existir um aumento.

```
1     if(effect=="l"){ //https://www.opencv-srf.com
          /2018/02/change-brightness-of-images-and-videos.
          html
2         cout<<"To increase light insert a positive <
              value> to decrease insert a negative <value
              >\n";
3         int value;
4         cout<<"choose value: "<<endl;
5         cin>>value;
6         Mat imageBrighnessHigh;
7         if(value>0){
8             image.convertTo(imageBrighnessHigh, -1, 1,
                  value); //increase the brightness
9             imshow("light_increase",imageBrighnessHigh);
10            waitKey(0);
11        }
12
13        Mat imageBrighnessLow;
14        if (value<0){
15            image.convertTo(imageBrighnessLow, -1, 1,
                  value); //decrease the brightness
16            imshow("light_decrease",imageBrighnessLow);
17            waitKey(0);
18        }
19
20
21        return 0;
22    }
```

Listing 2.5: Código fonte em C++ para aumentar ou diminuir a intensidade de luz da imagem



Aumento da Intensidade



Diminuição da Intensidade

```
utilizador@utilizador-Legion-Y540-15IRH-PG0:~/Desktop/Universidade/IC_AULAS/proj2/op
cv-example-src$ ../opencv-example-bin/ex_2 teste l
To increase light insert a positive <value> to decrease insert a negative <value>
choose value:
-90
```

Exemplo de compilação da diminuição da intensidade

Capítulo 3

Parte II

3.1 Exercício 3

Neste exercício, foi nos pedido para implementar uma classe para uma codificação de *Golomb*. Esta permite gerar um conjunto de códigos de tamanho variável livres de prefixo que podem ser utilizados na compressão de dados.

O código Golomb é uma família de códigos que dependem de um parâmetro inteiro, m $>$ 0 onde,

$$m = \left\lceil \frac{-1}{log\alpha} \right\rceil \tag{3.1}$$

A codificação envolve separar um inteiro em duas partes: a parte unária e a parte binária. Dado que queremos codificar um número inteiro, n, este é representado por 2 números, q e r:

$$q = \left\lfloor \frac{n}{m} \right\rfloor \tag{3.2}$$

$$r = n - qm \tag{3.3}$$

O quoeficiente q pode ter valores 0, 1, 2, 3... e dessa forma é representado pelo seu código unário correspondente.

O resto da divisão, r, pode assumir valores entre 0 e m-1 e irá representar o respetivo código binário.

Esta situação, descrita anteriormente, é válida para quando o valor de m é uma potência de 2. Se não o for, o processo anteriormente descrito não é valido, sendo o método alternativo apresentado de seguida.

- Começamos por definir um valor b que nos diz quantos bits serão necessários para a parte binária da codificação Golomb.

$$b = \lceil log_2 m \rceil \tag{3.4}$$

- Se $r < 2^b - m$, o valor de de r é codificado em binário usando b-1 bits.
- Caso contrário, é codificado o valor $r + 2^b - m$ em binário com b bits.

```
int Golomb::encode(int n){
    n = fold(n);
    int q = floor((int)(n / m));
    int r = n - (q * m);

    int nbitsbin;  // stores the number os bits of the
        binary part
    int value;     // value to be coded in the binary part

    // Verify if m is a power of two
    if ((m & (m-1)) == 0 && m != 0) {
        nbitsbin = b;
        value = r;
    }
    else{
        int x = pow(2, b) - m;
        // Verify if r is smaller than x
        if (r < x){
            nbitsbin = b - 1;
            value = r;
        }
        else{
            nbitsbin = b;
            value = r + x;
        }
    }
```

Listing 3.1: Código fonte em C++ Classe Golomb: Codificar

Um dos atributos desta classe vai ser um objeto da classe Bitstream, desenvolvida no projeto anterior, a qual é utilizada para escrever as codificações ou então lê-las no caso da descodificação.

O processo da descodificação envolve ler bits escritos num ficheiro. Começa-se por contar o número de 0's (do código unário). Tendo o número de 0's descobrimos o valor da parte unária (U) que será depois multiplicado por m. De seguida o processo terá duas alternativas dependendo se o valor de m for ou não uma potência de 2.

Caso seja, serão lidos b bits que corresponderão ao valor de R, sendo o valor descodificado obtido através da soma de R com o da parte unária multiplicada por m (mU + R).

Caso o valor de m não seja uma potência de 2, serão lidos b - *1* bits e dependendo do valor lido haverão duas possibilidades para o cálculo do valor descodificado. Se este valor for inferior a x, tendo em conta que x=$(2^b - m)$, R será igual a este valor e o valor descodificado será obtido da mesma forma que foi descrita na situação anterior (mU + R).

Por outro lado se o valor lido for superior a x, será lido mais um bit, sendo o valor de R atualizado com este valor e o valor descodificado será dado pela expressão anterior subtraída pelo valor de x ((mU + R) - x).

```
int Golomb::decode(){
    int U = 0;  // number of bits of the unary part
    int R = 0;  // number of bits of the rest
    int bit;    // current bit

    while(true){
        bit = file.readBit();
        if(bit == 1) break;
        U++;
    }
    // cout << "U= " << U << endl;
    // Verify if m is a power of two
    if ((m & (m-1)) == 0 && m != 0) {
        // vector<int> bin = file.readNbits(b); //reads the
            binary part
        int aux = b-1;
        for(int i = b-1; i >= 0; i--){
            bit = file.readBit();
            // cout << bit<< endl;
            if (bit != 0){
                R += pow(2, aux);
            }
            aux--;
        }
        // cout << "R= " << R << endl;
        return unfold(m*U + R);
    }
    else {
        vector<int> bin = file.readNbits(b-1); //read b-1
            bits from the binary part
        // cout << "b= " << b << endl;
        int aux = b-2;

        for(int i = 0; i < b-1; i++){
            if (bin[i] != 0){
                R += pow(2, aux);
```

```
35              }
36              aux--;
37          }
38
39          int x = (pow(2, b) - m);
40          if(R < x){
41              // cout << "R= " << R << endl;
42              return unfold(m*U + R);
43          }
44          else{
45              R = 2 * R; // "shift" of the bits that have been
                    read
46              int Lsbit = file.readBit();
47              if (Lsbit == 1){
48                  R++;
49              }
50              // cout << "x= " << x << endl;
51              // cout << "R= " << R << endl;
52              return unfold((m*U + R) - x);
53          }
54      }
55  }
```

Listing 3.2: Código fonte em C++ Classe Golomb: Descodificar

De forma a permitir a codificção de valores negativos, utilizamos um método em que os números negativos são representados por números ímpares positivos e os números positivos são representados pelos pares. Esses métodos são representado pelas funções *fold* e *unfold* da classe *Golomb*.

Para testar esta classe desenvolvemos o programa de teste test_golomb.cpp, que codifica valores positivos e negativos e que de seguida procede à descodficação dos mesmos de forma a verificar se existiram erros em algum destes processos.

Neste programa são também testadas funções auxiliares que foram desenvolvidas com vista a facilitar o processo de codificação de ficheiros de áudio e imagem.

Para além disso foi também desenvolvido o programa test_golomb2.cpp que testa a função auxiliar que irá servir para codificar e descodificar os headers dos ficheiros de aúdio.

Capítulo 4

Parte III

4.1 Exercício 4

Preditores

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

$$\begin{cases} \hat{X}_n^{(0)} = 0 \\ \hat{X}_n^{(1)} = x_{n-1} \\ \hat{X}_n^{(2)} = 2x_{n-1} - x_{n-2} \\ \hat{X}_n^{(3)} = 3x_{n-1} - 3x_{n-2} + x_{n-3} \end{cases} \tag{4.1}$$

Após o cálculo especificado no seguinte sistema de equações, faltava calcular o valor dos residuais que é dado por:

$$r_n = x_n - \hat{x}_n \tag{4.2}$$

Na reconstrução dos valores obtidos pelos preditores realiza-se a operação inversa:

$$x_n = r_n - \hat{x}_n \tag{4.3}$$

Logo, é necessário calcular $\hat{x}_n$ que é dado pelos preditores que mencionámos anteriormente. No entanto, em vez de serem utilizados os valores originais das samples, vão ser usados os valores dos residuais obtidos.

Para conseguirmos obter uma entropia mais baixa, desenvolvemos o nosso código de forma a que o processamento de som seja feito canal a canal, isto é, o preditor está dividido em 2 uma parte para o canal esquerdo e outra para o canal direito.

Portanto, o preditor faz as suas previsões para o canal esquerdo baseando-se na informação do canal esquerdo e faz as suas previsões no canal direito baseando-se na informação do canal direito.

No entanto, os residuais resultantes desta previsão de cada canal são colocados no mesmo vetor.

Lossless Codec

Para testar este codec é necessário compilar e correr o ficheiro test_audioCodec.cpp com um argumento de entrada que é o ficheiro a ser comprimido. Ao executar o ficheiro irá ser apresentado um prompt ao utilizador onde ele irá escolher o tipo de codec a usar. Neste caso, irá escolher a opção correspondente a lossless e de seguida o modo do preditor pretendido.

Para a implementação deste codec é necessário recolher os dados referentes ao ficheiro de áudio que serão precisos para que a reconstrução do ficheiro original no lado do descodificador seja possível. Logo, para esse efeito foi criada uma função adicional na classe Golomb que permite a codificação do cabeçalho que vai conter informação crucial na reconstrução do ficheiro original.

A informação contida num cabeçalho de áudio inclui o valor m que será utilizado pela classe Golomb, os campos associados ao ficheiro de áudio (nº de samples, sampleRate, formato e nº de canais), de seguida o modo do preditor utilizado, o modo do codificador (lossy ou lossless) e, por fim, o campo opcional que apenas existirá caso o codificador esteja em modo lossy que representa o número de bits a descartar no processo de quantização.

Golomb m	32 bits
Número de Samples	32 bits
Sample Rate	32 bits
Format	32 bits
Channels	4 bits
PredMode	2 bits
Lossy	1 bits
shQuant	5 bits

O valor de cada sample é enviado para o preditor que, por sua vez, irá calcular um valor que será utilizado para obter os residuais, r. Obtendo os residuais (r) calculamos o m ideal para codificar os residuais com códigos de Golomb. O método escolhido para calcular este valor é apresentado de seguida:

$$mean = \frac{\sum fold(r_n)}{N} \tag{4.4}$$

$$m = \left\lceil \frac{-1}{log_2\left(\frac{mean}{mean|1.0}\right)} \right\rceil \tag{4.5}$$

Após ser calculado o m, são criados os cabeçalhos e escritos no ficheiro. Para terminar o processo são escritos no ficheiro destino os valores dos residuais codificados através de códigos de Golomb . Isto é feito através de chamadas sucessivas à função *encode* da classe Golomb.

Por sua vez, o processo descodificação é executado através das operações opostas às que foram descritas anteriormente. Este processo é iniciado ao ler primeiro o cabeçalho através da função desenvolvida na classe Golomb. Ao ler o cabeçalho obtemos o m e outros dados que nos permitem reconstruir o ficheiro original

Tendo processado esta informação procedemos à leitura e descodificação de todos os valores dos residuais calculados. Essa leitura é feita com sucessivas chamadas à função decode da classe Golomb.

Para finalizar, após estes processos, vão ser copiados todos os dados obtidos no processamento de informação para um ficheiro àudio de output com recurso aos métodos da classe *libsndfile*.

4.2 Exercício 5

Lossy Codec

Para testar este codec é necessário compilar e correr o mesmo ficheiro indicado anteriormente: test_audioCodec.cpp com um argumento de entrada que é o ficheiro a ser comprimido. Aqui o utilizador irá escolher o codec lossy escolhendo de seguida o número de bits a serem descartados no processo de quantização e também o modo do preditor a utilizar.

Neste caso, a estrutura base é semelhante à descrita para o codec lossless. A alteração efetuada foi a adição de um processo de quantização aos residuais calculados com o preditor. E para isso foi também necessário alterar o preditor deste codec pois o cálculo dos residuais tem de refletir esta quantização que é realizada.

Assim sendo, sempre que é 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 direção oposta o mesmo número de vezes).

O processo de descodificação é semelhante ao realizado no codec lossless. A diferença vai de encontro à necessidade de reconstruir a quantização que foi efetuada. Essa reconstrução é feita efetuando um shift na direção oposta o mesmo número de vezes que foi realizada na codificação.

Resultados

Tendo em conta que,

$$Compression\ ratio = \frac{Uncompressed\ File}{Compressed\ File} \tag{4.6}$$

$$Space\ saving = 1 - \frac{Uncompressed\ File}{Compressed\ File} \tag{4.7}$$

apresentamos nas tabelas seguintes os resultados de vários testes efetuados aos codec de áudio. Desta forma iremos conseguir analisar e retirar conclusões sobre o desempenho e resultados obtidos dos codecs (primeiramente do codec lossless e de seguida do lossy).

Lossless:

As colunas das tabelas representam, respetivamente, o modo do preditor utilizado, a taxa de compressão, o espaço salvo através da compressão do ficheiro e também a duração do processo de compressão.

FILE: sample03.wav

PredMode	CompRatio	SpaceSaving(%)	Duration(s)
1	1.466236	31.798133	6.211558
2	1.525929	34.466134	5.995765
3	1.500377	33.350083	6.418600

FILE: sample04.wav

PredMode	CompRatio	SpaceSaving(%)	Duration(s)
1	1.366147	26.801439	4.516673
2	1.549826	35.476615	3.888854
3	1.617674	38.182829	3.590187

FILE: sample05.wav

PredMode	CompRatio	SpaceSaving(%)	Duration(s)
1	1.549500	35.463032	5.826299
2	1.725942	42.060627	5.498499
3	1.747565	42.777514	5.627314

FILE: sample06.wav

PredMode	CompRatio	SpaceSaving(%)	Duration(s)
1	1.778635	43.777119	5.836242
2	2.255081	55.655695	5.065854
3	2.549906	60.782871	4.417672

Analisando os resultados obtidos para os diferentes ficheiros de áudio, é possível concluir que em geral, o preditor 3 obteve os melhores resultados tanto a nível de taxa de compressão como a nível de duração do processo de compressão, o que significa que, para além de ter a capacidade de comprimir mais os ficheiros também o consegue fazer de forma mais eficiente.

Lossy (PredMode = 2):

Neste caso as colunas das tabelas representam, respetivamente, o número de bits a descartar no processo de quantização, a taxa de compressão, o espaço salvo através da compressão do ficheiro, a duração do processo de compressão e também o SNR (Signal-to-noise ratio) obtido ao comparar o ficheiro restaurado (após a compressão) com o seu original.

FILE: sample03.wav

BitsQuant	CompRatio	SpaceSaving(%)	Duration(s)	SNR(dB)
1	1.686861	40.718293	5.427951	71.9986
2	1.885766	46.971161	5.430606	63.5509
3	2.137650	53.219667	4.718977	56.5616
4	2.466756	59.460922	4.452133	50.1023
5	2.911542	65.653938	3.525287	43.863
7	4.318189	76.842145	2.328813	31.6593
9	6.557065	84.749275	1.518348	19.5806
11	10.94079	90.859892	0.839566	7.48592
13	13.454509	92.567548	0.728113	-4.91724

É possível concluir que conforme o número de bits que serão removidos no processo de quantização aumenta, a taxa de compressão, por sua vez, também aumenta, tal como seria esperado. Isto deve-se ao facto de serem considerados cada vez menos bits para representar cada amostra o que levará a uma poupança de espaço necessário para armazenar o ficheiro comprimido.

Conseguimos obervar que, quantos mais bits são removidos menor será o valor do SNR. Isto justifica-se pelo facto de que quando os bits são removidos são introduzidos erros na fonte de informação e daí surge um aumento de ruído, que consequentemente levará a que o valor do SNR diminua. Para o último valor do número de bits removidos, o ficheiro descomprimido apresenta quase só ruído.

Entropia (Lossy e lossless encoding, PredMode = 3)

Na tabela abaixo observamos a entropia associada ao ficheiro original e de seguida a obtida após o processo de predição dos codecs (lossless e lossy) para o caso do preditor 3, dado que, foi este que apresentou melhores resultados.

Audio File	Entropy (original)	Entropy (residuals, lossless)	Entropy (residuals, lossy)
sample03.wav	13.2678	10.441	8.46562
sample04.wav	14.1619	9.85233	7.85467
sample05.wav	12.4516	8.68766	6.83448
sample06.wav	11.8349	6.05798	4.16605

Para ambos os codecs conseguimos concluir que o preditor atingiu o seu objetivo. A sua finalidade era obter uma sequência de valores cuja entropia fosse menor que a sequência de valores original, isto é, menos símbolos binários necessários para codificar a fonte de informação.

Por fim, conclui-se também que entropia associada aos residuais obtidos através do preditor do codec lossy foi ligeiramente menor que a do codec lossless. O facto de a entropia ser menor implica, consequentemente, uma maior taxa de compressão o que conseguimos verificar pelos resultados obtidos anteriormente.

Para executar o programa para os processos de codificação lossless e lossy:

Executar programa: ../opencv-example-bin/test_audiocodec <input file>

E de seguida, deve ser selecionado o modo do codificador pretendido (0 para lossless e 1 para lossy), posteriormente é selecionado o modo do preditor (1, 2 ou 3) e, por fim, caso o codificador esteja em modo lossy é também pedido ao utilizador que este insira o número de bits a descartar no processo de quantização (valor entre 1 e 15).

Capítulo 5

Parte IV

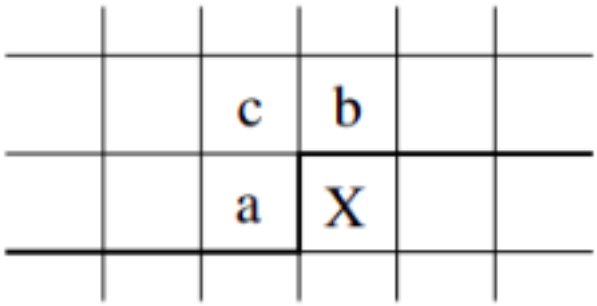
5.1 Exercício 6

Preditor

O preditor usado para este codec foi não linear e as suas equações são descritas como indicamos a seguir:

$$\hat{x} = \begin{cases} min(a,b)\ if\ c \geq\ max(a,b) \\ max(a,b)\ if\ c \leq\ min(a,b) \\ a + b - c\ otherwise \end{cases} \tag{5.1}$$

A base deste preditor é a mesma da configuração espacial do JPEG:



Configuração espacial do JPEG

Lossless image

A estrutura deste codec é baseada na estrutura que foi descrita no exercício 4. Por isso, o preditor utilizado já foi descrito na secção anterior. A diferença deste codec para o de aúdio está no processamento.

A transformação rgb <-> YCbCr usada foi 8-bit full-range é descrita na imagem abaixo:

$$\begin{bmatrix} Y \\ C_b \\ C_r \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} \quad \begin{matrix} Y \in [0, 255] \\ C_b \in [0, 255] \\ C_r \in [0, 255] \end{matrix} \tag{F.15}$$

The conversion from YCbCr to RGB is found by inserting the same weights into Eqs. F.11,F.12, andF.13:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1.000 & 0.000 & 1.403 \\ 1.000 & -0.344 & -0.714 \\ 1.000 & 1.773 & 0.000 \end{bmatrix} \cdot \begin{bmatrix} Y \\ C_b - 128 \\ C_r - 128 \end{bmatrix} \quad \begin{matrix} R \in [0, 255] \\ G \in [0, 255] \\ B \in [0, 255] \end{matrix} \tag{F.16}$$

Fórmulas usadas para conversão RGB <-> YCbCr

De seguida, é preciso manipular o tamanho das componentes Cr e Cb, porque estas necessitam serem *sub-sampling* do tipo 4:2:0.

As componentes Cb e Cr são 1/4 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.

Resumidamente, cada bloco de entrada é convertido do espaço de cores RGB para o espaço de cores YCbCr. A transformação usada usa um valor de entrada RGB com cada componente no intervalo [0 - 255] e transforma-o em Y, Cb e Cr, nos intervalos [0, 0, 255, 0], [-128, 0, 127, 0] e [-128, 0, 127, 0], respetivamente. O componente Y é deslocado para baixo em 128, de modo que também caia na faixa [-128, 0, 127, 0].

Para o cálculo do m do Golomb, usámos a média ponderada dos residuais após a operação *fold* em vez da média aritmética pois foram obtidos melhores resultados com este tipo de média. Todo o restante processo de cálculo foi semelhante ao utilizado para os ficheiros de áudio.

Resultados

```
FILE: baboon.ppm
Duration: 0.873509 seconds
Compression ratio: 2.6637
Space saving: 62.4582%
Entropy Original File: 7.69313
Entropy of the obtained residuals after prediction: 5.89652
```

Baboon Encode

```
FILE: monarch.ppm
Duration: 1.01313 seconds
Compression ratio: 3.77051
Space saving: 73.4784%
Entropy Original File: 7.61696
Entropy of the obtained residuals after prediction: 3.74381
```

Monarch Encode

Baboon original image

Restored_image

monarch original image

Restored_image

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

Executar Programa../opencv-example-bin/test_imageCodec <inputfile>

Contribuições dos autores

O trabalho foi dividido entre os três, sendo as percentagens para cada um as seguintes:

Bruno Silva -> 35%

Marta Oliveira -> 35%

Mariana Silva -> 30%

Webgrafia

https://elearning.ua.pt/pluginfile.php/3743066/mod_resource/content/4/ic-notas.pdf

https://stackoverflow.com/questions/14907964/mirror-image-in-opencv

https://www.opencv-srf.com/2018/02/change-brightness-of-images-and-videos.html

https://stackoverflow.com/questions/32190494/what-is-the-vec3b-type