

Universidade de Aveiro

## Informação e Codificação



Bruno Lemos 98221, Tiago Marques 98459, João  
Viegas 98372

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>1</b>
<b>2</b>	<b>Parte I</b>	<b>3</b>
2.1	Exercício 1 - Copy image . . . . .	3
2.2	Exercício 2 - Filter image . . . . .	3
2.2.1	a) Negative . . . . .	4
2.2.2	b)a) Mirrored - horizontal . . . . .	4
2.2.3	b)a) Mirrored - vertical . . . . .	4
2.2.4	c) Rotate . . . . .	5
2.2.5	d) Increases / Decreases . . . . .	5
<b>3</b>	<b>Parte II</b>	<b>7</b>
3.1	Exercício 3 - Golomb Class . . . . .	7
3.1.1	Encoder . . . . .	7
3.1.2	Decoder . . . . .	8
<b>4</b>	<b>Parte III</b>	<b>9</b>
4.1	Exercício 4 - <i>Lossless</i> áudio codec . . . . .	9
4.1.1	Detalhes de Implementação . . . . .	10
4.1.2	Resultados . . . . .	11
4.2	Exercício 5 - <i>Lossy</i> áudio coding . . . . .	12
4.2.1	Resultados . . . . .	13
<b>5</b>	<b>Parte IV</b>	<b>14</b>
5.1	Exercício 6 - Lossless Image codec . . . . .	14
5.1.1	Encoder e Decoder . . . . .	14
5.1.2	Validação e Resultados . . . . .	15

# Listas de Figuras

2.1	Resultado da copia pixel por pixel do airplane.ppm(original) para aviao.ppm(copia) . . . . .	3
2.2	Resultado da aplicação do filtro de cores negativa, à esquerda original à direita o negativo. . . . .	4
2.3	Resultado da aplicação do filtro de espelho, à esquerda original à direita a imagem espelhada. . . . .	4
2.4	Resultado da aplicação do filtro de espelho, à esquerda original à direita a imagem espelhada. . . . .	5
2.5	Resultado da aplicação do filtro de rotação, à esquerda original à direita a rodada por um múltiplo de 90. . . . .	5
2.6	Resultado da aplicação do filtro de aumento da intensidade da luz, à esquerda original à direita a imagem com intensidade aumentada. . . . .	6
2.7	Resultado da aplicação do filtro de aumento da intensidade da luz, à esquerda original à direita a imagem com intensidade diminuida. . . . .	6
3.1	Ficheiro original . . . . .	7
3.2	Ficheiro com os valores codificados através do codificador de Golomb . . . . .	8
3.3	Ficheiro original . . . . .	8
3.4	Ficheiro com os valores codificados através do codificador de Golomb . . . . .	8
4.1	Fluxo das operações sobre os dados quando ordem igual a 3 e o ficheiro áudio original possui 2 canais de áudio. . . . .	10
4.2	Exemplo do calculo da média. . . . .	11
4.3	Dados obtidos na compressão e descompressão de áudio usando o <i>codec</i> . . . . .	11
4.4	Dados obtidos na compressão e descompressão de áudio usando a opção <i>lossy</i> do <i>codec</i> com x=2000, y=1500 e ordem=3. . . . .	13
5.1	Como aplicar os modos de compressão numa imagem . . . . .	14
5.2	Imagen: Mode 8 . . . . .	15
5.3	Exemplo de comparação de hash's . . . . .	16
5.4	Compressão da imagem airplane.ppm através dos 8 modos de compressão . . . . .	16

5.5 Resultados do encoder e decoder . . . . .	17
---	----

# Capítulo 1

## Introdução

O presente relatório visa 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/brunolemos06/IC\\_Project2](https://github.com/brunolemos06/IC_Project2).

Para executar os comandos seguintes é necessário estar no diretório:

```
IC_Project2/src
```

Para compilar todos os programas

```
make
```

Para eliminar os executáveis

```
make clean
```

Executar o exercício 1

```
..../bin/ex1 <Input_Image_path> <Output_Image_path>
```

Executar o exercício 2a

```
..../bin/ex2 -a <Input_Image_path> <Output_Image_path>
```

Executar o exercício 2b

```
..../bin/ex2 -b <Input_Image_path> <Output_Image_path> <h/v>
```

Executar o exercício 2c

```
..../bin/ex2 -c <Input_Image_path> <Output_Image_path> <90/180/270>
```

Executar o exercício 2d

```
..../bin/ex2 -d <Input_Image_path> <Output_Image_path> <inc/decr>
```

Executar o exercício 3(Primeira Implementação)

```
..../bin/secgolomb <encode/decode> <m> <toencode/decoded_file> <decoded/toencode_file>
```

Executar o exercício 3(Segunda Implementação)

```
..../bin/golomb_codec_tests <encode/decode> <m> <toencode/decoded_file> <decoded/toencode_file>
```

Executar o exercício 4

Encode: ..../bin/golomb\_codec\_tests encode <to\_encode.wav> <encoded\_samples\_out.txt>

Opt. encode args: <-order [2,3]> <-x [1, num samples of file]> <-y [1, xl]>

Decode: ..../bin/golomb\_codec\_tests decode <encoded\_samples\_in.txt> <decoded\_out.wav>

Executar o exercício 5

**Encode:** .../bin/golomb\_codec\_tests encode <to\_encode.wav> <encoded\_samples\_out.txt> <-lossy [1,15]>  
Opt. encode args: <-order [2,3]> <-x [1, num samples of file]> <-y [1, x]>  
**Decode:** .../bin/golomb\_codec\_tests decode <encoded\_samples\_in.txt> <decoded\_out.wav>  
Executar o exercício 6

**Encode:**

```
..../bin/encoder_image <image.ppm> <out> [ mode=8,x=2500,y=2500 ]  
..../bin/encoder_image <image.ppm> <out> <mode> [ x=2500,y=2500 ]  
..../bin/encoder_image <image.ppm> <out> <mode> <X> <Y>
```

**Decode:**

```
..../bin/decoder_image <out> <out.ppm> [ mode=8,x=2500,y=2500 ]  
..../bin/decoder_image <out> <out.ppm> <mode> [ x=2500,y=2500 ]  
..../bin/decoder_image <out> <out.ppm> <mode> <X> <Y>
```

Importante notar que os exercícios compilados de cada programa são guardados na pasta **bin** e o código fonte na pasta **src**.

# Capítulo 2

## Parte I

Esta parte é dedicada à manipulação de imagem.

### 2.1 Exercício 1 - Copy image

Neste exercício executamos o programa ex1 onde vai ser criada uma cópia pixel por pixel da imagem original num novo ficheiro.

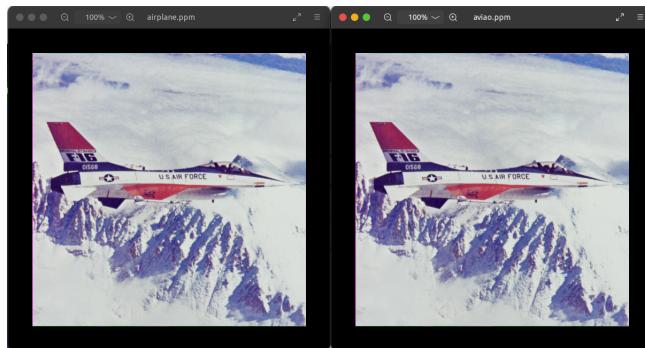


Figura 2.1: Resultado da copia pixel por pixel do airplane.ppm(original) para aviao.ppm(copia).

### 2.2 Exercício 2 - Filter image

Nos exercícios a baixo realizamos alguns filtros de imagem aplicados pixel a pixel. Os filtros aplicados foram feitos alterando as entradas de cores de cada pixel.

### 2.2.1 a) Negative

Nesta alínea aplicámos um filtro de cores negativo à imagem original pixel a pixel. A cada uma das entradas dos pixels atribuímos o valor da média entre os três valores de entrada original.

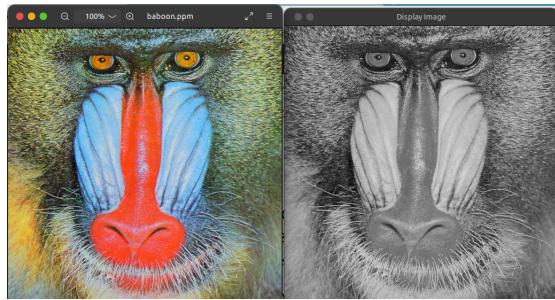


Figura 2.2: Resultado da aplicação do filtro de cores negativa, à esquerda original à direita o negativo.

### 2.2.2 b)a) Mirrored - horizontal

Nesta alínea aplicamos um filtro de espelho pixel a pixel. Considerando i o valor da linha e j o valor da coluna é trocado o valor de  $(i,j)$  pelo valor do pixel da linha  $i$  e da coluna  $n\text{colunas}-j-1$ .

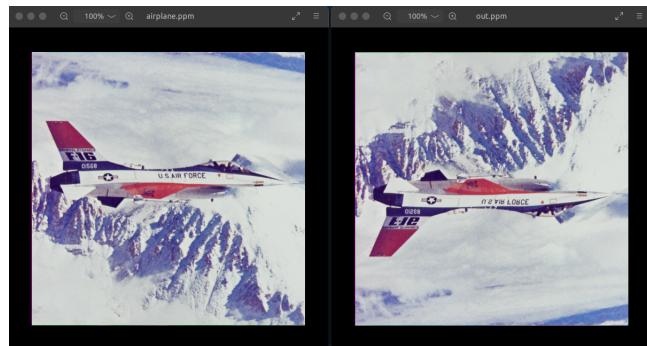


Figura 2.3: Resultado da aplicação do filtro de espelho, à esquerda original à direita a imagem espelhada.

### 2.2.3 b)a) Mirrored - vertical

Nesta alínea aplicamos um filtro de espelho pixel a pixel. Considerando i o valor da linha e j o valor da coluna é trocado o valor de  $(i,j)$  é trocado pelo pixel da linha  $n\text{linhas}-i-1$ .

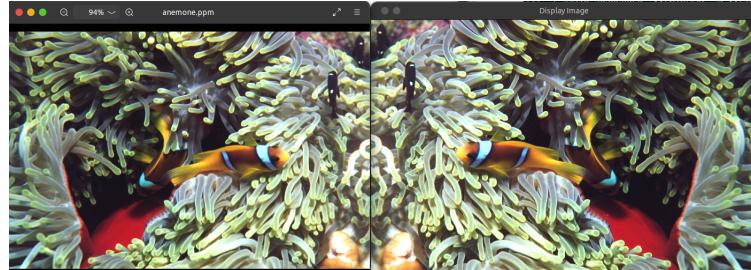


Figura 2.4: Resultado da aplicação do filtro de espelho, à esquerda original à direita a imagem espelhada.

#### 2.2.4 c) Rotate

Nesta alínea aplicámos uma rotação à imagem original pixel a pixel, podendo rodar a imagem num valor múltiplo de 90. Se o valor for 90 cada valor de entrada de cada pixel  $(i,j)$  é trocado pelo pixel da linha  $j$  e coluna  $n\text{colunas}-i-1$ . Se o valor for 180 cada valor de entrada de cada pixel  $(i,j)$  é trocado pelo pixel da linha  $n\text{linhas}-i-1$  e coluna  $n\text{colunas}-j-1$ . Se o valor for 270 cada valor de entrada de cada pixel  $(i,j)$  é trocado pelo pixel da linha  $n\text{linhas}-j-1$  e coluna  $i$ .

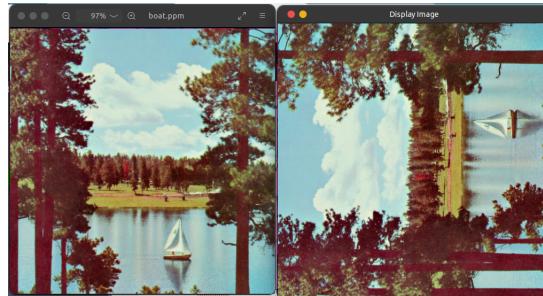


Figura 2.5: Resultado da aplicação do filtro de rotação, à esquerda original à direita a rodada por um múltiplo de 90.

#### 2.2.5 d) Increases / Decreases

Nesta alínea aplicamos um filtro para aumento e diminuição da intensidade da luz da imagem pixel a pixel. Para isso, a cada valor de cor do pixel é incrementado 50 ou decrementado 50.

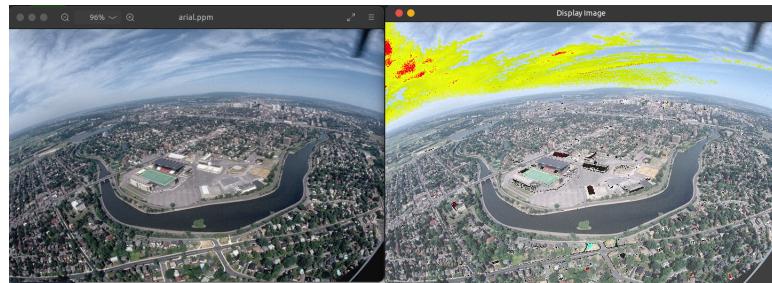


Figura 2.6: Resultado da aplicação do filtro de aumento da intensidade da luz, à esquerda original à direita a imagem com intensidade aumentada.

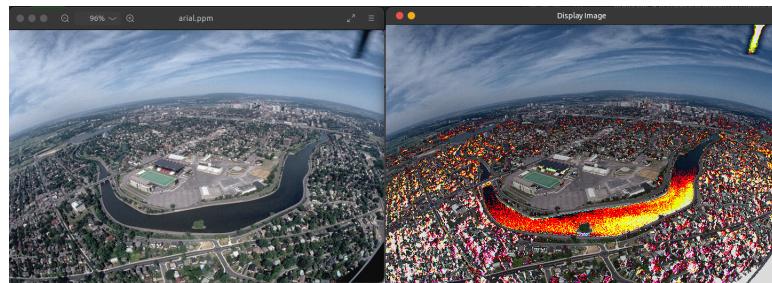


Figura 2.7: Resultado da aplicação do filtro de aumento da intensidade da luz, à esquerda original à direita a imagem com intensidade diminuida.

# Capítulo 3

## Parte II

### 3.1 Exercício 3 - Golomb Class

Neste exercício realizamos a class de golomb de duas formas diferentes. Na primeira que vamos apresentar a baixo calculamos o quociente e o valor de resto de cada um dos valores a codificar. De seguida na mesma palavra representamos o quociente em código unário e o resto em código binário. Por ultimo adicionamos um bit 0 ou 1 se o valor for negativo ou positivo, respetivamente. O valor de m é passado na chamada da função. Na segunda implementação é feita um mapeamento, no caso da existência de valores negativos a codificar, em que os valores negativos passam a ser valores positivos pares ( $x^*-2$ ) e os valores positivos passam a ser valores positivos ímpares ( $(x^2) +1$ ). De seguida, calculamos o quociente do valor em código unário e o resto da sua divisão em código binário. O valor de m é passado na inicialização da classe.

#### 3.1.1 Encoder

No encoder é passado um ficheiro com valores inteiros 1,2,3,4,5... na chamada da função. No output vão ser escritos os valores codificados com um m=5, como podemos ver nas figuras a baixo.

```
encode.txt
1 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Figura 3.1: Ficheiro original

```
1 0000001001000110001110100001001010100101100101110110000110010110100110110011011101110000
```

Figura 3.2: Ficheiro com os valores codificados através do codificador de Golomb

### 3.1.2 Decoder

No decoder é passado um ficheiro com valores codificados com um  $m=5$  separados por espaços na chamada da função. No output vão ser escritos os valores descodificados com um  $m=5$ , como podemos ver nas figuras a baixo.

```
1 0000001001000110001110100001001010100101100101110110000110010110100110110011011101110000
```

Figura 3.3: Ficheiro original

```
≡ encode.txt  
1 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
```

Figura 3.4: Ficheiro com os valores codificados através do codificador de Golomb

# Capítulo 4

## Parte III

Neste exercício é nos pedido a implementação de um *codec* utilizando a classe de *golomb* previamente implementada, foi usada a segunda implementação, com o objetivo de comprimir e descomprimir áudio.

### 4.1 Exercício 4 - *Lossless* áudio *codec*

A função de codificação de áudio recebe inicialmente um ficheiro *.wav* para descodificar, um parâmetro *x* ( $x > 0$ ), que serve para atualizar o valor de *m* de *x* em *x* valores, um parâmetro *y* ( $x \geq y > 0$ ), que utiliza os *y* valores anteriores para calcular o novo *m*, um parâmetro para a ordem (2 ou 3) que vai definir como se calcula a previsão dos valores e, finalmente, um nome para o ficheiro resultante da compressão.

O primeiro passo começa pela leitura do ficheiro *.wav*. Após esta leitura é criado um *header*, que contém toda a informação necessária para realizar o processo de descodificação. Para o processo de codificação os primeiros (ordem \* numero de canais do áudio) valores mantém os seus valores originais e são mapeados, nos restantes valores são calculados os erros, realizando uma subtração do valor previsto ao valor real. De seguida, é realizado um mapeamento desse erro, para um numero positivo par para quando o erro  $< 0$  ou então para um numero positivo ímpar quando o erro  $\geq 0$ . Por fim todos os valores são codificados e escritos num ficheiro. O *header* é a primeira coisa a ser escrita no ficheiro e só depois são escritos os valores codificados.

Enquanto o processo dos cálculos dos erros decorre, o valor de *m* vai sendo alterado de *x* em *x* valores, em que *y* valores dos erros mapeados anteriores são utilizados para calcular a média e consequentemente o novo valor de *m*.

Quanto à função de descodificação de áudio, esta função recebe um ficheiro codificado e o nome do ficheiro que será criado com o áudio descodificado. Inicialmente a função lê o *header* que irá conter toda a informação desnecessária para realizar a descodificação do ficheiro. Consequentemente o processo de descodificação realiza as operações do processo de codificação mas por ordem inversa.

### 4.1.1 Detalhes de Implementação

Calculo do Erro:

Primeiro é preciso saber que ordem aplicar para o calculo do valor previsto, possuímos 2 opções:

Formula da ordem 3:

$$\hat{X}_n = 3.X_{n-1} - 3.X_{n-2} + X_{n-3}$$

Formula da ordem 2:

$$\hat{X}_n = 2.X_{n-1} - X_{n-2}$$

Depois de sabia a nossa ordem é preciso calcular o valor previsto, nesta implementação esse valor é calculado usando apenas os valores reais de cada canal, ou seja, para cada calculo de valor previsto só são usados valores reais daquele canal. E por fim é calculado o erro, usando a formula:

$$Err = X_n - \hat{X}_n$$

Mapeamento: Se  $Err < 0 \Rightarrow Err = Err \cdot (-2)$  senão  $Err = (Err+1) \cdot 2$   
 O mapeamento neste caso é realizado fora da classe *golomb* devido à necessidade de calcular o m dinamicamente.  
 E por fim são codificados os erros.

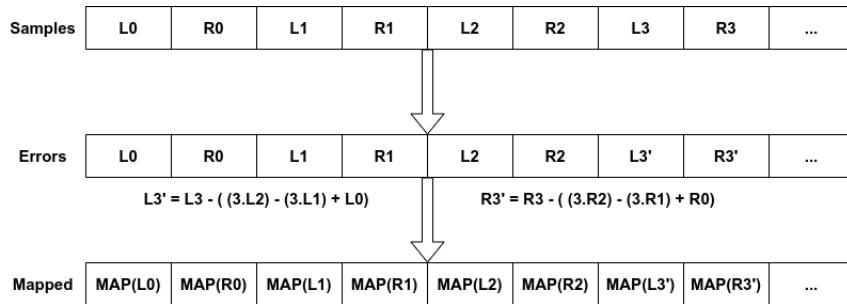


Figura 4.1: Fluxo das operações sobre os dados quando ordem igual a 3 e o ficheiro áudio original possui 2 canais de áudio.

Calculo dinâmico de m:

O parâmetro de x no programa será inversamente proporcional à frequência do calculo dinâmico de m, ou seja, se x for um valor alto a frequência com que vamos alterar o valor de m é mais baixa do que se usarmos um valor de x mais pequeno.

O parâmetro de y no programa define o intervalo de valores usados para calcular

a média de valores que será usada para calcular o novo m, regra geral, quanto maior o valor de y maior será o espaço de amostra e consequentemente melhor será a 'qualidade' do novo m calculado.

Exemplo do calculo da média:

*Nota:* Os valores usados são os valores mapeados dos erros.

**x = 20, y = 5**

VALUE_12	VALUE_13	VALUE_14	VALUE_15	VALUE_16	VALUE_17	VALUE_18	VALUE_19	VALUE_20

$$\text{medium} = \text{sum}(\text{VALUE\_16} \dots \text{VALUE\_20}) / y$$

Figura 4.2: Exemplo do calculo da média.

Formula do calculo de m:

$$m = \lceil \frac{-1}{\log_2 \frac{\text{medium}}{\text{medium}+1}} \rceil$$

#### 4.1.2 Resultados

order	ulate m every x sam the values of the pr	encode time(s)	decode time(s)	size(bytes)	sample.wav size
3	2000	1500	0.28551	0.225974	1649417
3	2000	1250	0.279312	0.233027	1647526
3	2000	750	0.272588	0.23008	1642397
3	200	150	0.2758	0.235092	1587947
3	200	125	0.275282	0.228388	1588009
3	200	75	0.271255	0.230363	1588565
2	2000	1500	0.265338	0.223366	1615384
2	2000	1250	0.278489	0.222707	1614479
2	2000	750	0.277038	0.231149	1612585
2	200	150	0.273595	0.226857	1592042
2	200	125	0.272867	0.235988	1592319
2	200	75	0.275034	0.228474	1593257
order	ulate m every x sam the values of the pr	encode time(s)	decode time(s)	size(bytes)	sample06.wav size
3	2000	1500	0.416099	0.230564	1616718
3	2000	1250	0.406595	0.22945	1616793
3	2000	750	0.40386	0.24449	1618838
3	200	150	0.416804	0.237283	1634383
3	200	125	0.43266	0.235526	1637226
3	200	75	0.420764	0.243571	1644943
2	2000	1500	0.431836	0.235086	1702610
2	2000	1250	0.414558	0.23137	1702609
2	2000	750	0.416253	0.238914	1705652
2	200	150	0.416534	0.23768	1731575
2	200	125	0.425797	0.243722	1737261
2	200	75	0.418645	0.243855	1753586

Figura 4.3: Dados obtidos na compressão e descompressão de áudio usando o codec.

Analizando primeiramente a ordem do *prediction* (valor da primeira coluna), podemos observar que para ordem igual a 3 obtemos uma maior compressão mas também demoramos mais tempo a des/codificar, este aspeto torna-se mais visível para ficheiros de maior tamanho. O inverso irá acontecer para uma ordem igual a 2, ou seja, uma compressão menor em comparação com a ordem igual a 3 mas mais rápida. Através da terceira ordem obtemos uma maior compressão devido a esta 'olhar' mais para o passado (usa 3 valores anteriores) do que a segunda ordem (usa 2 valores anteriores) no cálculo do valor 'futuro'.

Olhando agora para os valores relacionados com a atualização dinâmica do valor de m, podemos ver que quanto maior for a frequência da atualização de m (causado por um menor valor de x) e quanto maior for o intervalo de valores para a média (maior valor de y), que está diretamente relacionado com o cálculo do valor de m, maior será a compressão mas consequentemente, quanto menor o valor de x e maior o valor de y mais será o tempo de des/codificação.

Quanto as percentagens de compressão, para o ficheiro *sample.wav* obtemos:  
Maior taxa de compressão:

$$\frac{\text{UncompressedSize}}{\text{CompressedSize}} = \frac{2116844}{1587947} = 1,3331$$

Menor taxa de compressão:

$$\frac{\text{UncompressedSize}}{\text{CompressedSize}} = \frac{2116844}{1649417} = 1,2834$$

Quanto as percentagens de compressão, para o ficheiro *sample06.wav* obtemos:  
Maior taxa de compressão:

$$\frac{\text{UncompressedSize}}{\text{CompressedSize}} = \frac{4235996}{1630557} = 2,5979$$

Menor taxa de compressão:

$$\frac{\text{UncompressedSize}}{\text{CompressedSize}} = \frac{4235996}{1753586} = 2,4156$$

*Nota:* Será possível obter uma melhor compressão variando os valores de x e y.

A discrepância entre as taxas de compressão entre os dois ficheiros deve-se ao tipo de som guardado nos ficheiros, o *sample.wav* possui um som com vários instrumentos musicais e o ficheiro *sample06.wav* embora possua um som vibrante não tem um volume tão alto, nem tantas fontes de som como o ficheiro anterior.

## 4.2 Exercício 5 - *Lossy* áudio *coding*

Neste exercício é nos pedido uma opção *lossy*, com perda de dados, para o *codec*. Para ativar essa opção na execução do *codec*, basta passar o comando

com '-lossy [1, 15]'. Para realizar a quantização dos valores no decorrer do processo de codificação todas os valores das *samples* sofrem um *shift right* de n bits e no processo de descodificação acontece o processo inverso, ou seja, um *shift left* de n bits dos valores já descodificados e retornados ao seu valor 'original' (*desmapeados*).

#### 4.2.1 Resultados

number of bits cut	encode time(s)	decode time(s)	size(bytes)	SNR(dB)	Maximum Absolute Error	sample06.wav size
2	0.362168	0.184509	1181006	57.804	3	4235996
4	0.319955	0.138586	891913	44.3165	15	
8	0.290123	0.107296	751059	19.8287	255	
12	0.244833	0.085567	580027	-4.56603	4095	

Figura 4.4: Dados obtidos na compressão e descompressão de áudio usando a opção *lossy* do *codec* com  $x=2000$ ,  $y=1500$  e ordem=3.

Ao analisar os resultados obtidos podemos observar, que quanto mais quantização o som sofrer menos tempo será necessário no processo de des/codificação mas em contrapartida maior será o ruído introduzido no áudio, visível através da observação dos valores do *signal-to-noise ratio (SNR)* quando se compara o ficheiro de áudio original com o ficheiro de áudio descodificado. Focando agora na coluna '*Maximum Absolute Error*' podemos verificar que o processo de quantização está a decorrer corretamente visto que ao retirar n bits o erro máximo absoluto por *sample* que ocorre é o de valor  $2^n$ .

# Capítulo 5

## Parte IV

### 5.1 Exercício 6 - Lossless Image codec

Neste exercício vamos explicar como procedemos, aplicámos e verificamos o nosso lossless image codec.

#### 5.1.1 Encoder e Decoder

O nosso encode tem 8 modos de compressão e após vários testes verificámos que o modo 8 é o que tem melhor compressão em grande parte dos casos.

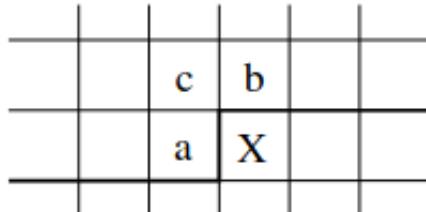


Figura 5.1: Como aplicar os modos de compressão numa imagem

Para cada pixel da imagem é calculado 3 vezes este modo porque a cada pixel é lhe atribuído 3 valores. ( Por exemplo:  $a = (180,0,255)$  ).

Os modos de compressão foram os seguintes:

- Mode 1 ->  $X = a$
- Mode 2 ->  $X = b$
- Mode 3 ->  $X = c$

- Mode 4 ->  $X = a+b-c$
- Mode 5 ->  $X = a(b-c)/2$
- Mode 6 ->  $X = b+(a-c)/2$
- Mode 7 ->  $X = (a+b)/2$
- Mode 8 ->

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

Figura 5.2: Imagem: Mode 8

Foi calculado para todos os valores da imagem o valor *predicted*, excepto a primeira coluna e a primeira linha que foram valores reais da imagem. Foi subtraído o valor real pelo valor que passa pelo cálculo dos modos.

Para diferenciar valores positivos e negativos decidimos transformar todos os valores negativos em valores positivos pares e valores positivos ímpares:

**value é negativo :**  $-2 * value$

**value é positivo :**  $2 * value + 1$

Para uma melhor compressão foi atribuído um M dinâmico à função do algoritmo do *codec*, baseado classe *Golomb*. Para tal temos um parâmetro de entrada no nosso codificador em que temos um X e um Y, o seu efeito é que a cada X iterações dos valores já calculados calcula um novo valor de M com base nos últimos Y valores, **Importante** :  $x \geq y$  . Após vários testes usando *scripts* conseguímos atribuir um melhor valor *default* de X,Y, esse valor é 2500 para ambos. O valor *default* do modo de codificação é o modo 8. Todo este valor codificado é guardado numa string e assim guardado num ficheiro em binário através de uma função *write\_bin\_to\_file*.

O processo do *decoder* é exatamente o inverso de todo este processo mencionado a cima.

### 5.1.2 Validação e Resultados

Como é um *codec lossless* não pode existir quaisquer perdas de valores atribuídos aos pixeis. Para tal foi calculado a *hash* do ficheiro original e comparado à *hash* do ficheiro descodificado, e foi assim verificado 0% de perdas.

A taxa de compressão foi calculada através da seguinte fórmula :

$$\text{Compression} = \frac{\text{EncodedFile}}{\text{OriginalFile}}$$

```

~/Projects/UA-ECT/4ano/IC/IC_Project2/src
(18:51:48 on main)→ ./bin/encoder_image ../Images/airplane.ppm out
Colors written : 786432
Taxa de Compressão : 0.581292
Execution time : 0.162873 seconds
~/Projects/UA-ECT/4ano/IC/IC_Project2/src
(18:51:59 on main)→ ./bin/decoder_image out out.ppm
Encoded size in bytes: 457147
Encoded size in Mbytes: 0.435969
Execution time : 0.103544 seconds
~/Projects/UA-ECT/4ano/IC/IC_Project2/src
(18:52:15 on main)→ md5sum ../Images/airplane.ppm out.ppm
06fbb1b7fc90bb8ae0d048638db4a99a  ../Images/airplane.ppm
06fbb1b7fc90bb8ae0d048638db4a99a  out.ppm

```

Figura 5.3: Exemplo de comparação de hash's

Os valores dos modos foram testados em várias imagens e o modo 8 foi o que obteve melhores resultados. Na imagem abaixo um exemplo de compressão através dos 8 modos de compressão:

airplane	
mode	compressão
1	0,632
2	0,632
3	0,673
4	0,605
5	0,594
6	0,598
7	0,597
8	0,581

Figura 5.4: Compressão da imagem airplane.ppm através dos 8 modos de compressão

Importante nestes resultados é analisar a taxa de compressão e também o tempo de computação quer do encoder quer do decoder. Quanto melhor estes valores melhor o codificador.

Imagens	compressão	tempo de computação	
		encoder	decoder
airplane	0,58	0,19	0,1
anemone	0,67	0,23	0,14
arial	0,77	0,28	0,16
baboon	0,81	0,19	0,12
bike3	0,69	0,47	0,3
boat	0,72	0,21	0,12
girl	0,61	0,17	0,29
house	0,61	0,06	0,03
lena	0,63	0,19	0,12
monarch	0,57	0,23	0,14
peppers	0,67	0,18	0,11
tulips	0,6	0,25	0,16
Média	0,65	0,2s	0,13s

Figura 5.5: Resultados do encoder e decoder

Comparámos também os nossos resultados através de compressores mais conhecidos.

# Contribuições 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:

- Bruno Lemos - 33.3%
- Tiago Marques - 33.3%
- João Viegas - 33.3%