

Informação e Codificação

Projeto 1

Univerisade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática

Ana Rosa (98678), Sara Gonçalves (98376), Julia
Abrantes

<https://github.com/JuliaAbrantes/IC-projeto1>



2022/2023

Conteúdo

1	Introdução	1
2	Parte I	2
2.1	Exercício 2	2
2.2	Exercício 3	3
2.3	Exercício 4	4
2.4	Exercício 5	5
3	Parte II	6
3.1	Exercício 6	6
3.2	Exercício 7	7
4	Parte III	9
4.1	Exercício 8	9
5	Contribuição dos Autores	10

Lista de Figuras

2.1	Histogramas canal 0 e 1, respetivamente	3
2.2	Histogramas canal MID e SIDE, respetivamente	3
2.3	Histogramas canal 0 e 1 com 9 bits de resolução, respetivamente	4
2.4	Histogramas canal 0 e 1 com 5 bits de resolução, respetivamente	4
2.5	Histogramas canal 0 e 1 com 5 bits de resolução, respetivamente	5
3.1	Encoder e Decoder teste 1	7
3.2	Encoder e Decoder teste 2	8

Capítulo 1

Introdução

Este relatório visa descrever a resolução do projeto 1, no âmbito da unidade curricular de Informação e Codificação.

O software desenvolvido encontra-se disponível em: <https://github.com/JuliaAbrantes/IC-projeto1>.

No diretório onde se encontram os ficheiros desenvolvidos, encontra-se um ficheiro README a exemplificar como executar todo o código desenvolvido. Além disso, foram adicionados diretórios onde se encontram os resultados dos testes (histogramas e ficheiros de áudio) para cada exercício.

Capítulo 2

Parte I

2.1 Exercício 2

Este exercício tem como objetivo alterar a classe *WAVHist* dada, de modo a fornecer também o histograma da média dos canais (canal **MID**) e o histograma da diferença dos canais quando o áudio é stereo, isto é, quando contém dois canais (canal **SIDE**).

O canal **MID** (mono) contém informação idêntica em ambos os canais (esquerdo e direito), ao contrário do canal **SIDE** (stereo) que contém informação que difere entre os canais, esquerdo e direito.

Para o canal **MID**, calculamos a média dos canais esquerdo e direito, usando a expressão:

$$\frac{L+R}{2}$$

Para o canal **SIDE** foi usada a expressão:

$$\frac{L-R}{2}$$

No sentido de obter as amostras dos canais esquerdo e direito, foi implementada a função `separateSamples` no ficheiro [wav_hist.h](#). Esta função apenas coloca todas as amostras do canal esquerdo num vetor e todas as amostras do canal direito noutro vetor. Assim, é possível aplicar diretamente a fórmula nas funções `updateMIDchannel` e `updateSIDEchannel` e colocar os valores obtidos num vetor.

No caso do canal **MID** os valores são inseridos no vetor `midChannel` e no caso do canal **SIDE** os valores são colocados no vetor `sideChannel`.

Para a visualização gráfica dos histogramas, foi usado a aplicação externa GNUPLOT, sendo que no eixo das abcissas estão representados os valores das amostras (samples) e no eixo das coordenadas é representado o número de ocorrências. Na figura 2.1 é possível observar os histogramas do canal 0 e 1, respectivamente. Na figura 2.2 são representados os histogramas do canal MID e SIDE, respectivamente. É de notar que foi utilizado o ficheiro de áudio sample01.wav disponível no diretório audioFiles.

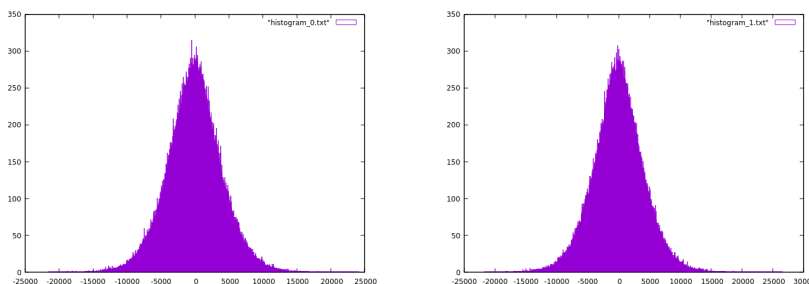


Figura 2.1: Histogramas canal 0 e 1, respetivamente

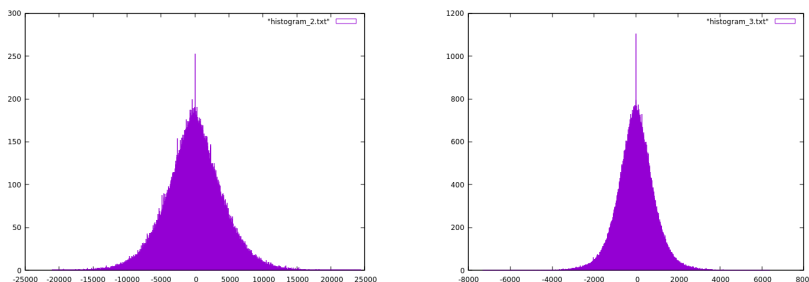


Figura 2.2: Histogramas canal MID e SIDE, respetivamente

2.2 Exercício 3

O exercício 3 tem como objetivo reduzir a quantidade de bits de resolução do ficheiro áudio, isto é, diminuir os bits usados para representar cada amostra. Nesse sentido, o ficheiro [wav_quant.cpp](#) e [wav_quant.h](#) foram desenvolvidos. O ficheiro .h implementa a função de quantização e o ficheiro .cpp aplica essa função a um ficheiro de áudio.

Nas figuras abaixo, aplicou-se ao ficheiro de áudio sample01.wav, que foi o utilizado também no exercício 2, uma quantização de 9 bits de resolução (figura 2.3) e 5 bits de resolução (2.4). Assim, é possível verificar a diferença que se obtém ao reduzir os bits para representar as amostras.

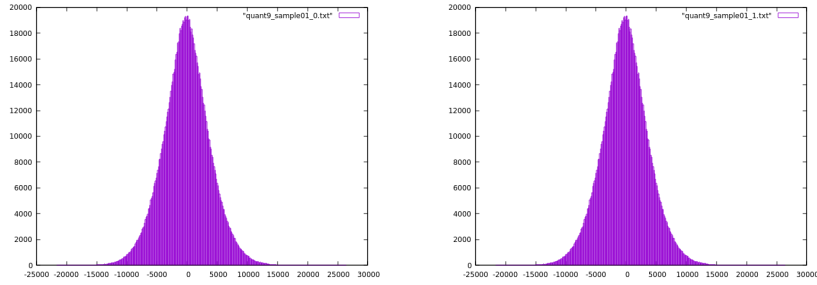


Figura 2.3: Histogramas canal 0 e 1 com 9 bits de resolução, respetivamente

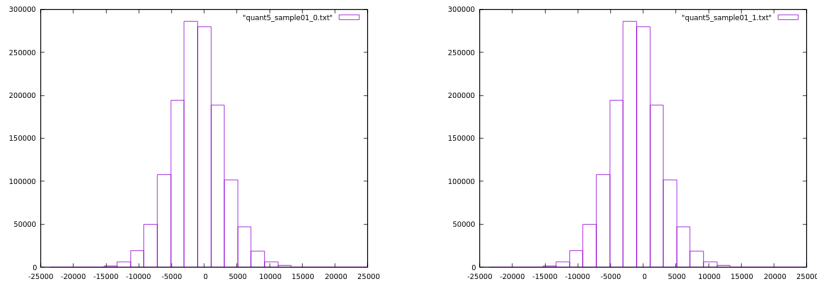


Figura 2.4: Histogramas canal 0 e 1 com 5 bits de resolução, respetivamente

2.3 Exercício 4

Neste exercício foi criado o programa [wav_cmp.cpp](#). Este exercício é importante para ter em conta que a operação de quantização efetuada no exercício 3 introduz erros. Nesse sentido, é necessário determinar o *signal-to-noise ratio* do ficheiro de áudio quantizado em relação ao ficheiro original, bem como o erro absoluto máximo por amostra.

O **SNR** é definido pela expressão:

$$SNR = 10 * \log_{10} * \frac{E_x}{E_r} (dB) \quad (2.1)$$

Sendo que E_x é a energia do sinal, calculada como a soma de todas as amostras do ficheiro original ao quadrado, e E_r a energia do ruído, calculada como a soma da diferença entre a amostra original e a amostra comprimida ao quadrado.

O Erro absoluto máximo por amostra é calculado através do módulo da diferença entre a amostra original e a amostra comprimida.

Na figura abaixo, verifica-se que o ficheiro de audio (sample01.wav) quantizado com 5 bits tem um erro absoluto por amostra bastante maior do que o quantizado com 9 bits.

Relativamente ao SNR de cada ficheiro, ambos não são os melhores. No entanto, o valor do SNR para o ficheiro quantizado com 9 bits é considerado bom, pelo que o de 5 bits é o mínimo aceite para estabelecer uma ligação não fiável.

```
anaclaudiagcf:~/Documents/UA/4_Ano/IC/project_1/sndfile-example-src$ ../sndfile-example-bin/wav_cmp ../audioFiles/sample01.wav quant9_sample01.wav
Signal-to-noise ratio: 34.5089 dB
Maximum per sample absolute error: 127
anaclaudiagcf:~/Documents/UA/4_Ano/IC/project_1/sndfile-example-src$ ../sndfile-example-bin/wav_cmp ../audioFiles/sample01.wav quant5_sample01.wav
Signal-to-noise ratio: 19.3804 dB
Maximum per sample absolute error: 2047
anaclaudiagcf:~/Documents/UA/4_Ano/IC/project_1/sndfile-example-src$
```

Figura 2.5: Histogramas canal 0 e 1 com 5 bits de resolução, respetivamente

2.4 Exercício 5

No exercício 5 a classe WAVEffects foi criada no ficheiro [wav_effects.h](#). A classe possui 2 métodos públicos, ambos recebem uma sequência de frames, um atraso, um ganho e produzem um ficheiro de áudio com eco simples para o método simpEco

$$y(n) = x(n) + gain * x(n - delay) \quad (2.2)$$

ou múltiplo para o método multipEco.

$$y(n) = x(n) + gain * y(n - delay) \quad (2.3)$$

Capítulo 3

Parte II

Neste exercício o objetivo é o desenvolvimento de uma classe `BitStream`. A classe irá permitir ler/escrever de/para um ficheiro binário. Foram criados dois ficheiros, `BitStream.cpp` e `BitStream.h`. O ficheiro `BitStream.h` contém a implementação das funções (explicado na secção 3.1) e o ficheiro `BitStream.cpp` serve para testar as funções desenvolvidas na classe `BitStream` (exemplificado na secção 3.2).

3.1 Exercício 6

O ficheiro `BitStream.h` contém a seguinte implementação:

- `BitStream(char *fname, char m)`
Construtor que permite fazer a escolha entre a leitura ou escrita de um ficheiro binário.
- `void write_bit(int bit)`
Função que escreve um bit no ficheiro binário. Neste caso, teve de se ter em atenção que um ficheiro binário é composto por bytes, i.e, apenas se pode inserir no mínimo 1 byte de cada vez, logo não se pode simplesmente escrever um único bit. Pelo que, foi necessário criar uma variável "buffer" de um byte que só após ser preenchida com 8 bits é que é feita a escrita de 1 byte no ficheiro binário.
- `unsigned char read_bit()`
Função que lê um bit do ficheiro binário. Como referido na função anterior, o ficheiro binário é composto por bytes, logo também não se pode ler 1 bit de cada vez. Assim, através da variável "buffer", sempre que essa variável chegar aos 8 bits lidos (1 byte lido), lê-se a partir daí cada bit.
- `void write_Nbits(string bits)`

Função que escrever n bits no ficheiro binário. Esta função utiliza a função `write_Nbits`.

- vector <int> `read_Nbits(int nBits)`

Função que lê n bits do ficheiro binário. Esta função utiliza a função `read_bit`.

- unsigned int `binaryFile_size()`

Função que retorna a quantidade de bytes existentes no ficheiro binário.

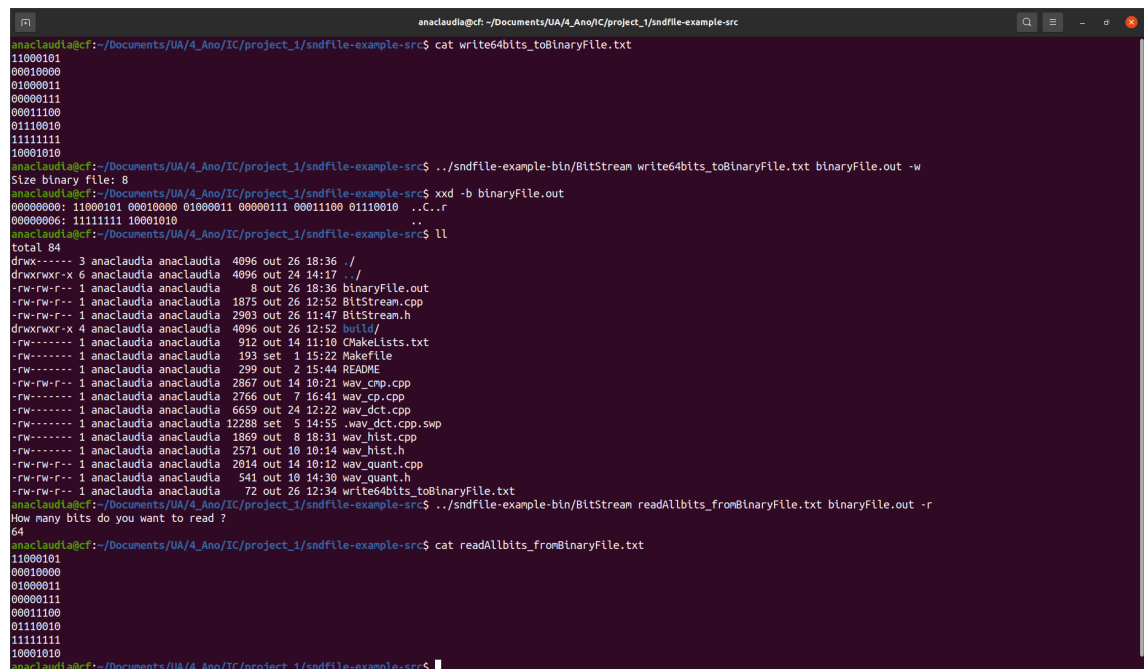
3.2 Exercício 7

O ficheiro `BitStram.cpp` implementa um decoder e um encoder.

Para testar o decoder é necessário no terminal ter '-r' como opção. O decoder baseia-se em ler n bits do ficheiro binário para um ficheiro de texto, sendo que o ficheiro de texto contém apenas valores 0's e 1's.

No caso do encoder, no terminal coloca-se a opção '-w'. O encoder escreve n bits (0's e 1's) de um ficheiro de texto para um ficheiro binário.

Os seguintes testes foram realizados:



```
anaclaudia@cf: ~/Documents/UA/4_Ano/IC/project_1/sndfile-example-src
anaclaudia@cf:~/Documents/UA/4_Ano/IC/project_1/sndfile-example-src$ cat write64bits_toBinaryFile.txt
11000101
00010000
01000011
00000111
00011100
01110010
11111111
10001010
anaclaudia@cf:~/Documents/UA/4_Ano/IC/project_1/sndfile-example-src$ ./sndfile-example-bin/BitStream write64bits_toBinaryFile.txt binaryFile.out -w
Size binary file: 8
anaclaudia@cf:~/Documents/UA/4_Ano/IC/project_1/sndfile-example-src$ xxd -b binaryFile.out
00000000: 11000101 00010000 01000011 00000111 00011100 01110010  ..C..r
00000006: 11111111 10001010
anaclaudia@cf:~/Documents/UA/4_Ano/IC/project_1/sndfile-example-src$ ll
total 84
drwxrwxr-x 3 anaclaudia anaclaudia 4096 out 26 18:36 ./
drwxrwxr-x 6 anaclaudia anaclaudia 4096 out 24 14:17 ../
-rw-rw-r-- 1 anaclaudia anaclaudia 8 out 26 18:36 binaryFile.out
-rw-rw-r-- 1 anaclaudia anaclaudia 1875 out 26 12:52 BitStream.cpp
-rw-rw-r-- 1 anaclaudia anaclaudia 2903 out 26 11:47 BitStream.h
drwxrwxr-x 4 anaclaudia anaclaudia 4096 out 26 12:52 build/
-rw-rw-r-- 1 anaclaudia anaclaudia 912 out 14 11:10 CMakelists.txt
-rw-rw-r-- 1 anaclaudia anaclaudia 193 set 1 15:22 Makefile
-rw-rw-r-- 1 anaclaudia anaclaudia 299 out 2 15:44 README
-rw-rw-r-- 1 anaclaudia anaclaudia 2867 out 14 10:21 wav_cmp.cpp
-rw-rw-r-- 1 anaclaudia anaclaudia 2766 out 7 16:41 wav_cp.cpp
-rw-rw-r-- 1 anaclaudia anaclaudia 6659 out 24 12:22 wav_dct.cpp
-rw-rw-r-- 1 anaclaudia anaclaudia 12288 set 5 14:55 .wav_dct.cpp.swp
-rw-rw-r-- 1 anaclaudia anaclaudia 1869 out 8 18:31 wav_hist.cpp
-rw-rw-r-- 1 anaclaudia anaclaudia 2571 out 10 10:14 wav_hist.h
-rw-rw-r-- 1 anaclaudia anaclaudia 2014 out 14 10:12 wav_quant.cpp
-rw-rw-r-- 1 anaclaudia anaclaudia 541 out 10 14:30 wav_quant.h
-rw-rw-r-- 1 anaclaudia anaclaudia 72 out 26 12:34 writes4bits_toBinaryFile.txt
anaclaudia@cf:~/Documents/UA/4_Ano/IC/project_1/sndfile-example-src$ ./sndfile-example-bin/BitStream readAllbits_fromBinaryFile.txt binaryFile.out -r
How many bits do you want to read ?
64
anaclaudia@cf:~/Documents/UA/4_Ano/IC/project_1/sndfile-example-src$ cat readAllbits_fromBinaryFile.txt
11000101
00010000
01000011
00000111
00011100
01110010
11111111
10001010
anaclaudia@cf:~/Documents/UA/4_Ano/IC/project_1/sndfile-example-src$
```

Figura 3.1: Encoder e Decoder teste 1

```

anacaudia@cf:~/Documents/UA/4_Ano/IC/project_1/sndfile-example-src$ ../sndfile-example-bin/BitStream readAllbits_fromBinaryFile.txt binaryFile.out -r
How many bits do you want to read ?
35
anacaudia@cf:~/Documents/UA/4_Ano/IC/project_1/sndfile-example-src$ cat readAllbits_fromBinaryFile.txt
11000101
00010000
01000011
00000111
000
anacaudia@cf:~/Documents/UA/4_Ano/IC/project_1/sndfile-example-src$ ../sndfile-example-bin/BitStream readAllbits_fromBinaryFile.txt binaryFile.out -r
How many bits do you want to read ?
80
80
The binary file has only 64 bits.
anacaudia@cf:~/Documents/UA/4_Ano/IC/project_1/sndfile-example-src$ cat write19bits_toBinaryFile.txt
10001100
11000000
111
anacaudia@cf:~/Documents/UA/4_Ano/IC/project_1/sndfile-example-src$ ../sndfile-example-bin/BitStream write19bits_toBinaryFile.txt binaryFile.out -w
Size binary file: 2
anacaudia@cf:~/Documents/UA/4_Ano/IC/project_1/sndfile-example-src$ 

```

Figura 3.2: Encoder e Decoder teste 2

Na figura 3.1 aplica-se em primeiro o encoder. Lê-se do ficheiro 'write64bits_toBinaryFile.txt' 64 bits e escreve-se os 64 bits no ficheiro binário. O comando 'xxd -b ficheiro_binário' mostra o conteúdo dos valores escritos no 'binaryFile.out'. Após isso, aplica-se o decoder, em que se lê desse mesmo ficheiro binário para um ficheiro de texto. Sendo que neste exemplo lê-se apenas 64 bits. Na figura 3.2, é exemplificado uma situação em que apenas se lê 35 bits e outra em que se pretende ler 80 bits (que dará erro, porque apenas se pode ler 64 bits).

Capítulo 4

Parte III

4.1 Exercício 8

Neste exercício, o ficheiro [wav_dct.cpp](#) foi reutilizado. Para aplicar o lossy encoder, todos os coeficientes DCT foram convertidos para binário de 16 bits e foram escritos num ficheiro binário através da função `write_Nbits` da classe `BitStream`. Para a aplicação do lossy decoder, foram lidos todos os bits do ficheiro binário (que representam os coeficientes DCT) através da função `read_Nbits` da classe `BitStream`.

Para efetuar o DCT inverso foi necessário converter todos os conjuntos de 16 bits para decimal com sinal. Para isso foi usada a função `binaryToSignedDecimal` que se encontra no ficheiro [wav_dct.h](#). Para finalizar, todos esses valores foram inseridos no vetor `x_dct_dec` e aplicados no DCT inverso.

Capítulo 5

Contribuição dos Autores

Todos os participantes deste projeto têm a mesma percentagem de participação.