UNIVERSIDADE DE AVEIRO

DEPARTAMENTO DE ELECTRÓNICA, TELECOMUNICAÇÕES E INFORMÁTICA

INFORMAÇÃO E CODIFICAÇÃO

PROJETO 1

Autores:
António Ferreira (89082)
Guilherme Claro (98432)
Luis Couto (89078)

https://github.com/Toja-Ferreira/IC$_G$07$_{Proj}$1

2022

Contents

1 Introdução

O presente relatório visa descrever a resolução do Projecto 1 desenvolvido no âmbito da unidade curricular de Informação e Codificação.

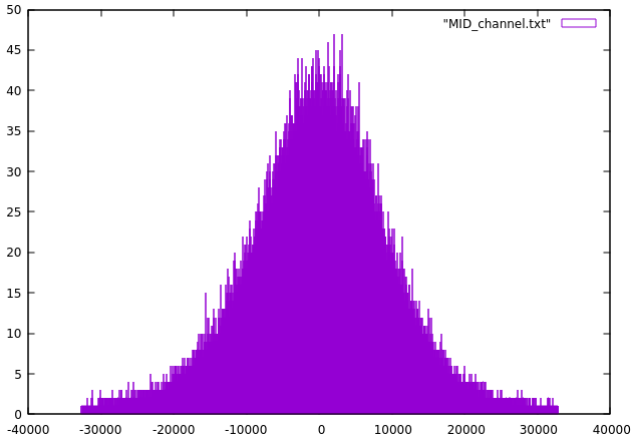
No directório base do projeto existe um ficheiro README.md com instruções de como compilar e testar todos os programas desenvolvidos.

2 Parte I

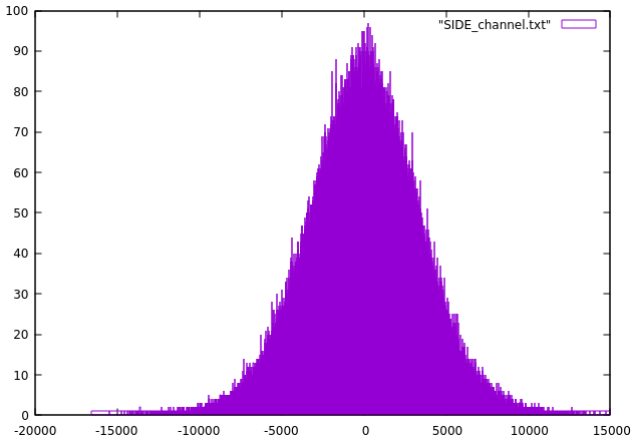
2.1 Exercício 2

Neste exercício foi nos fornecido um codigo que produzia um histograma do output do canal 0 e 1 (ou canal esquerdo e direito) e tinhamos como objetivo alterar o codigo para criar a saida da media dos canais (MID channel) através desta equação (L + R)/2, e tambem da diferença entre os canais (Side channel) com esta equação (L - R)/2

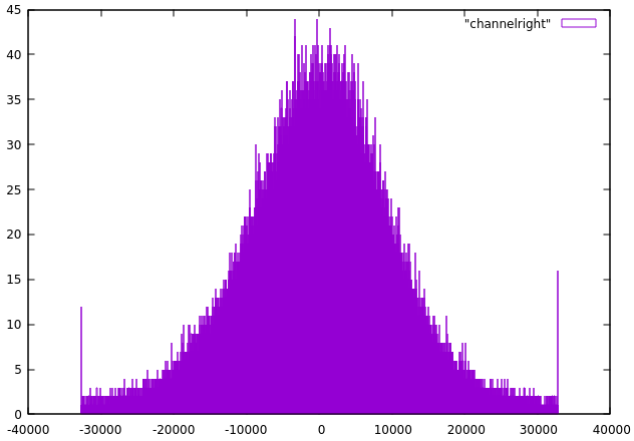
```
#Compilaçao
>../sndfile-example-bin/wav_hist sample.wav 0
#Ao abrir o gnuplot
>gnuplot: plot "MID_channel.txt" with boxes
```



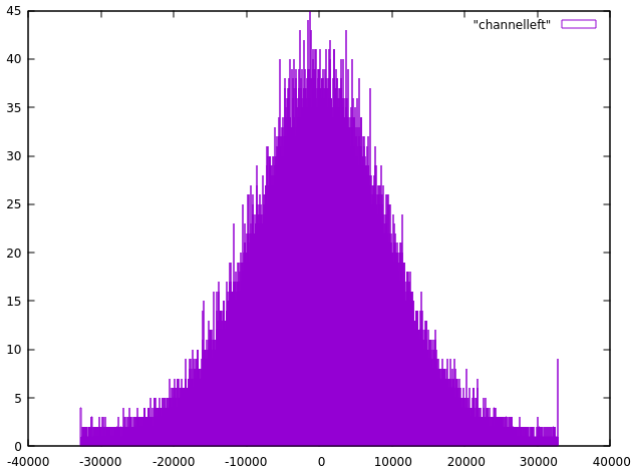

```
#Compilaçao
>../sndfile-example-bin/wav_hist sample.wav 0
#Ao abrir o gnuplot
>gnuplot: plot "SIDE_channel.txt" with boxes
```



```
#Passar para o output para ficheiro de texto
>../sndfile-example-bin/wav_cp sample.wav 1 > channelright
#Ao abrir o gnuplot
>gnuplot: plot "channelright.txt" with boxes
```




```
#Passar para o output para ficheiro de texto
>../sndfile-example-bin/wav_cp sample.wav 1 > channelleft
#Ao abrir o gnuplot
>gnuplot: plot "channelleft.txt" with boxes
```

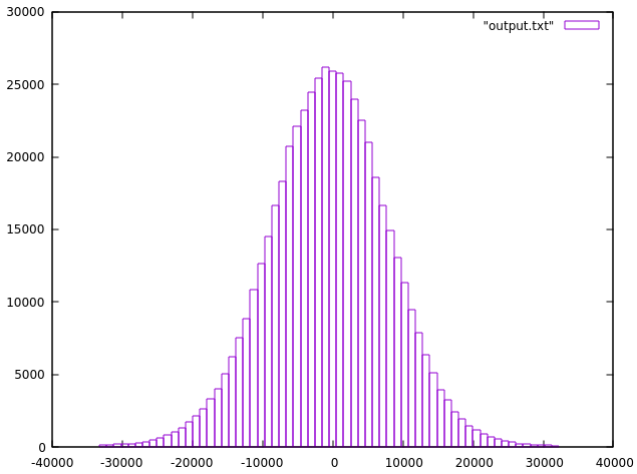


Para obtermos estes resultados alteramos o codigo do ficheiro `wav_hist.h` mas mais precisamente na função **void update** para obter output do aux e do s para podermos calcular o output do SIDE channel e do MID channel e converter para um ficheiro de texto. Tambem usamos o gnuplot para obter os histogramas(aconselhado pelos professores).

2.2 Exercício 3

Para este exercício foi implementado um programa que consegue reduzir o numero de bits para representar o ficheiro audio disponibilizado (**sample.wav**). Portanto ao fazer isto a representação no histograma vai apenas mostrar amostras em certos espaços de tempo, o que podemos ver na imagem abaixo

```
#Compilação
> ../sndfile-example-bin/wav_quant sample.wav 10 output.wav
```



2.3 Exercício 4

Neste programa foi nos pedido para construir um codigo que obtivesse o **SNR** (signal-to-noise ratio) de um audio "comprimido" em relação ao audio inicial , e o absoluto erro maximo por amostra.

Como audio comprimido para comparação usamos o audio **output.wav** do exercicio anterior já que foram usados menos bits.

Implementamos no codigo esta função para obtermos o resultado esperado.

$$\mathrm{SNR} = 10 \log_{10} \frac{E_x}{E_r} \quad \mathrm{dB} \ \text{(decibel)},$$

```
#Comando para compilar o progama
> ../sndfile-example-bin/wav_cmp sample.wav output.wav

#output obtido
SNR: 23.2637 dB
Maximum per sample absolute error: 1023
```

2.4 Exercício 5

Neste exercicio tinhamos de aplicar efeitos num ficheiro audio.

2.4.1 Single echo

Echo é um som ou sons causados pela reflexão das ondas numa superficie e que voltam para o ouvinte.

Para produzir um echo aplicamos esta função no nosso codigo:

$$y(n) = x(n) + ax(n-1).$$

```
#Produçao do echo pelo terminal
> ../sndfile-example-bin/wav_effects sample.wav out.wav single-echo
```

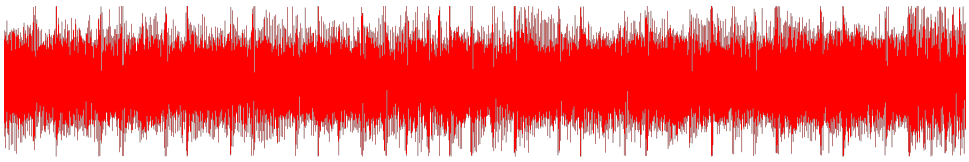


Figure 1: Waveform normal

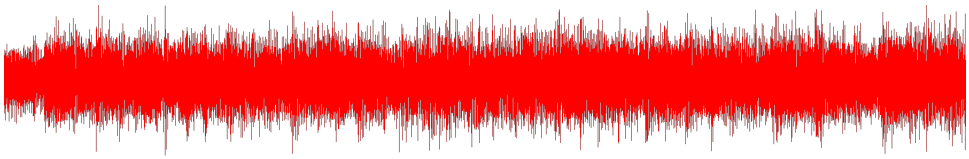


Figure 2: Waveform com echo

Para obter as imagens a cima usamos este website para converter os ficheiros wav para waveform link

3 Parte II

3.1 Exercício 6

A classe **BitStream** permite ler e escrever bits de um e num ficheiro binário, respetivamente.
A lógica fundamental desta implementação consiste no uso de um buffer e de um pointer, que assistem na leitura/escrita de bits.
Á medida que cada bit é escrito/lido do ficheiro, o buffer e o pointer são atualizados, de maneira a manter registo da posição de escrita/leitura. Dado que os ficheiros apenas podem ser escritos em bytes, ao fechar o ficheiro, caso esteja em modo de escrita e o pointer não aponte para o ultimo bit (bit 8), os restantes bits são escritos a 0.

```
BitStream();
BitStream(const char *filename, char modeIn);
void writeBit(char bit);
void writeNBits(const char *bitsToWrite, int numBitsToWrite);
auto readBit();
auto readNBits(int numBitsToRead);
void closeFile();
```

Exemplos do uso destas funções são os ficheiros `nBits_test.cpp` e `singleBit_test.cpp` que se encontram dentro da pasta `test_programs`.

3.1.1 Ler e Escrever 1 bit

No programa `singleBit_test.cpp` são testadas as funções:

```
// Função base do modo de escrita, recebe e escreve um bit no ficheiro.

void writeBit(char bit);
// Função base do modo de escrita, lê um bit do ficheiro.
auto readBit();
```

Para testar o programa (no terminal):

```
# Estando no diretório base do projeto
> cd Part_II/exe6/test_programs
> ./singleBit_test test_read_file

# Se o ficheiro for lido corretamente receberá a mensagem
File named single_write_file has been written!
File named test_read_file contains the following bits:
0010001110000000

# Comparando com o ficheiro original:
> xxd -b test_read_file
00000000: 00100011 10000000
```

3.1.2 Ler e Escrever N bits

No programa `nBits_test.cpp` são testadas as funções:

```
// Função que escreve N bits no ficheiro, faz uso da função de escrita
// de 1 bit até todos os bits indicados estarem escritos
void writeNBits(const char *bitsToWrite, int numBitsToWrite);

// Função que escreve N bits no ficheiro, faz uso da função de leitura
// de 1 bit até todos os bits indicados estarem lidos
auto readNBits(int numBitsToRead);
```

Para testar o programa (no terminal):

```
# Estando no diretório base do projeto
> cd Part_II/exe6/test_programs
> ./nBits_test test_read_file

# Se o ficheiro for lido corretamente receberá a mensagem
File named multiple_write_file has been written!
The first 16 bits from test_read_file are:
0010001110000000

# Comparando com o ficheiro original:
> xxd -b test_read_file
00000000: 00100011 10000000
```

3.2 Exercício 7

Usando a classe **BitStream** desenvolvida no exercício anterior, foram implementados dois programas, um codificador (**encoder.cpp**) e um descodificador (**decoder.cpp**). Estes dois programas serviram como testes suplementares da classe.

3.2.1 Encoder.cpp

Este programa converte um ficheiro de texto com 0s e 1s ao seu equivalente binário, onde cada byte do ficheiro binário criado representa oito dos bits do ficheiro de texto original.

Para testar o programa (no terminal):

```
# Estando no diretório base do projeto
> cd Part_II/exe7/test_programs
> ./encoder test.txt

# Se o ficheiro for codificado com sucesso receberá a mensagem:
Text file encoded successfully to encodedFile.

# Para ver o resultado:
> cat encodedFile
Hello World!

# Comparando o ficheiro original (texto) com o ficheiro codificado (binário):
> cat test.txt
01001000011001010110110001101100011011110010000
01010111011011110111001001101100011001000010001

> xxd -b encodedFile
00000000: 01001000 01100101 01101100 01101100 01101111 00100000  Hello
00000006: 01010111 01101111 01110010 01101100 01100100 00100001  World!
```

3.2.2 Decoder.cpp

Este programa faz o inverso do anterior, converte um ficheiro binário no seu ficheiro de texto equivalente.

Para testar o programa (no terminal):

```
# Estando no diretório base do projeto
> cd Part_II/exe7/test_programs
> ./decoder encodedFile

# Se o ficheiro for descodificado com sucesso receberá a mensagem:
Binary file decoded successfully to decodedFile.txt.

# Comparando o ficheiro original (texto) com o ficheiro descodificado (texto):
> cat test.txt
0100100001100101011011000110110001101111001000000
0101011101101111011100100110110001100100001000001

> cat decodedFile.txt
0100100001100101011011000110110001101111001000000
0101011101101111011100100110110001100100001000001
```

4 Parte III

4.1 Exercício 8

Neste exercício foi pedido a implementação de um lossy audio codec baseado na **DCT (Discrete Cosine Transform)**, com o audio é processado bloco a bloco. Cada bloco é convertido usando a **DCT**, os quoeficientes são apropriadamente quantizados e, usando a classe **BitStream**, os bits são escritos num ficheiro binário. Inversamente, o decoder utiliza este ficheiro para reconstruir uma versão aproximada do áudio original, comprimido, como mínimo de perdas de qualidade.

4.1.1 DCT

Na implementação deste programa, foram implementadas as seguintes fórmulas:

Relação direta (DCT-II), usada na codificação:

$$X(k) = \sqrt{\frac{2}{N}} \cdot a(k) \cdot \sum_{n=0}^{N-1} x(n) \cdot cos(\frac{\pi K(2n+1)}{2N}), k = 0, 1, ..., N-1$$

Relação inversa (IDCT), usada na descodificação:

$$x(n) = \sqrt{\frac{2}{N}} \cdot \sum_{k=0}^{N-1} a(k) \cdot X(K) \cdot cos(\frac{\pi K(2n+1)}{2N}), n = 0, 1, ..., N-1$$

onde

$$\alpha(0) = \frac{1}{\sqrt{2}}$$

e

$$\alpha(k) = 1, k \neq 0$$

Estas formulas foram implementadas nas funções:

```
void dctTransform(vector<double> &samples)
void dctInvert(vector<double> &samples)
```

Em termos de constantes relacionadas com estas transformações, foram tomadas as seguintes escolhas:

```
// Tamanho de cada bloco a ser processado no DCT
const size_t block_size = 1024;
// % de coeficientes de baixa frequencia a serem mantidos
const double dctFrac = 0.2;
// Numero de bits a serem cortados na quantização
int bit_cut = 4;
```

A codificação foi implementada com sucesso, sendo os dados do ficheiro de audio transformados usando DCT-II, quantizados e finalmente escritos num ficheiro binário. Durante a codificação, foram escritos nos primeiros bits do ficheiro binário dados relativos ao ficheiro de audio original (frames, samplerate, numero de canais e formato), que mais tarde serão usados na descodificação.

A descodificação foi parcialmente implementada, sendo os dados lidos do ficheiro binario, convertidos para os seus valores e escritos num novo ficheiro de audio criado. No entanto, não foi implementada a transformação destes dados usando o IDCT, pelo que o decoder funcional parcialmente.

Para testar o programa (no terminal):

```
# Estando no diretório base do projeto
> cd Part_III/exe8/test_programs
> ./test

# Se o ficheiro for codificado corretamente receberá a mensagem:
Encoding audio file into binary file, please wait...
A binary file with the encoded audio data has been created!

# Se o ficheiro for descodificado corretamente receberá a mensagem:
Decoding binary file into audio file, please wait...
The audio file has been created!
```

5 Contribuição dos autores

Todos os autores participaram no desenvolvimento deste projeto de forma igual, na qual a percentagem de cada membro fica:

- Antonio Ferreira - 33.33%
- Guilherme Claro - 33.33%
- Luis Couto - 33.33%