

Informação e Codificação

Projeto 1

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

30 de outubro de 2022

Índice

1	Introdução	1
2	Parte I	2
2.1	Exercício 2	2
2.2	Exercício 3	4
2.3	Exercício 4	5
2.4	Exercício 5	6
3	Parte II	9
3.1	Exercício 6	9
3.2	Exercício 7	11

Lista de Figuras

2.1	hist.dat	3
2.2	histM.dat	3
2.3	histS.dat	4
2.4	terminal	8

Capítulo 1

Introdução

O presente relatório tem como objetivo descrever a resolução do Projeto 1 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>

Para este projeto utilizamos a biblioteca *libsndfile* e o programa *gnuplot*. No ficheiro README.md no repositório estão as indicações de como compilar cada exercício.

Capítulo 2

Parte I

2.1 Exercício 2

Neste exercício era pedido para providenciar um histograma da média dos canais (mid channel) e a diferença de canais (side channel) quando o áudio é estéreo, isto é, com dois canais.

$$\begin{aligned}\text{MID CHANNEL} &= (\text{L} + \text{R})/2 \\ \text{SIDE CHANNEL} &= (\text{L} - \text{R})/2\end{aligned}$$

Para isso, criámos um programa que tem como argumentos de entrada um ficheiro de som (tipo wav) existente. Com o uso da biblioteca libsndfile conseguimos guardar a informação de cada canal num vetor. Para proceder à contagem de cada ocorrência de valores, recorreremos a uma estrutura de dados do tipo vetor que por sua vez usa um map em que as chaves (tipo short) são os valores lido do ficheiro de áudio e estão associadas ao número de ocorrências.

Finalmente, para apresentarmos a informação num histograma recorreremos ao gnuplot.

De seguida, mostramos os comandos necessários para verificar os resultados.

Como compilar o exercício 2: **make wav_hist**
Comando para executar o programa: **../sndfile-example-bin/wav_hist**
<input file> <channel>

O channel pode ser 0 or 1 para canal esquerdo ou direito respetivamente, s para side e m for mid.

Para abrir os histogramas:

```
gnuplot
plot "histM.dat"
plot "histS.dat"
```

plot "hist.dat"

No eixo das abscissas estão representados todos os valores das samples e no eixo das coordenadas está representado o número de ocorrências

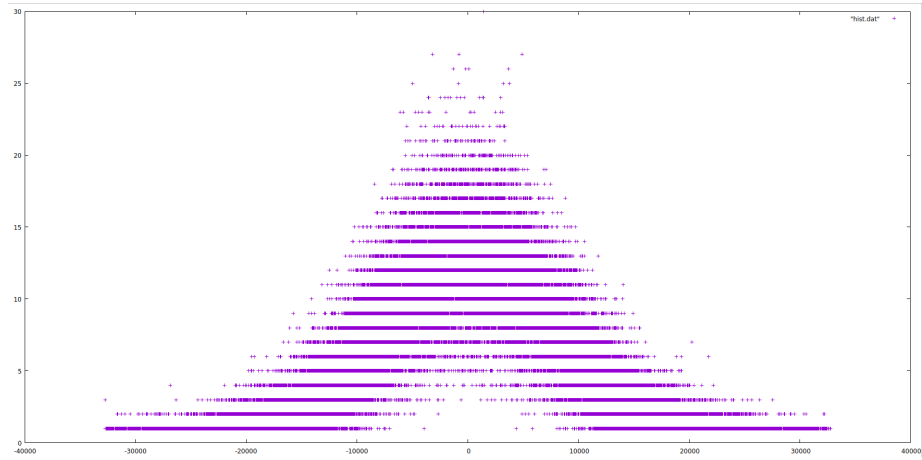


Figura 2.1: hist.dat

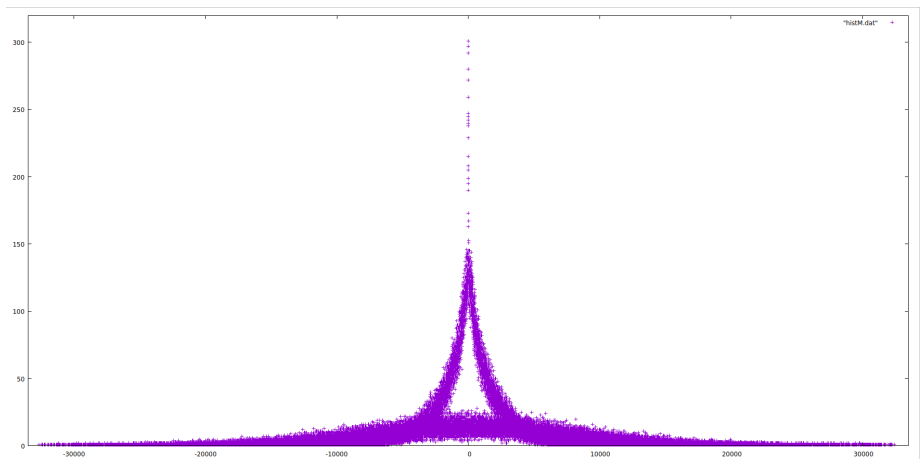


Figura 2.2: histM.dat

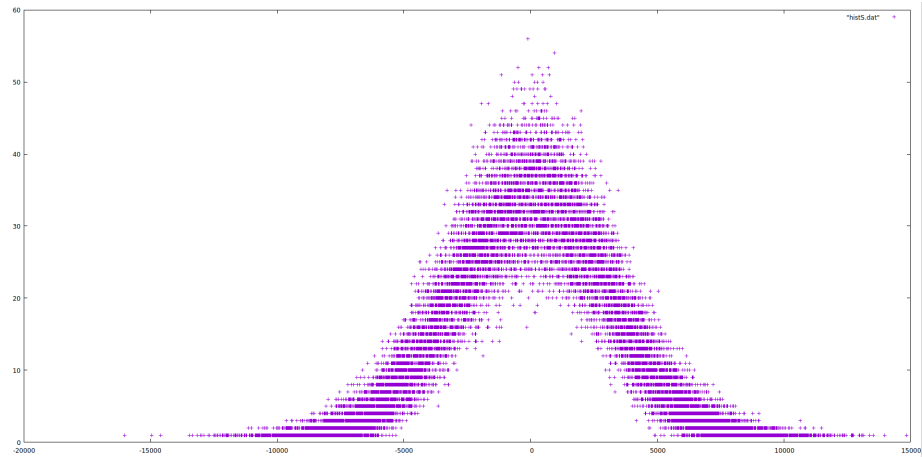


Figura 2.3: histS.dat

2.2 Exercício 3

O objetivo deste exercício era a quantização de som, isto é, reduzir o número de bits usados para representar cada sample de um ficheiro de áudio.

O processo de quantização de som representa a transformação de um sinal contínuo num sinal discreto. Esta técnica é irreversível, uma vez que implica perdas de informação.

Para este exercício, o utilizador irá inserir o ficheiro que quer quantizar, o nome do ficheiro onde irá ser guardado o resultado e o número de bits a descartar. Este valor tem de estar entre 1 a 16 bits pois os ficheiro WAV fornecidos são compostos por amostras de 16 bits.

Depois disso realizámos um deslocamento para a direita dos bits menos significativos da sample e, para não alterar o tamanho dela, recolocamos os bits a 0 realizando um descolamento para a esquerda. Desta forma, foi reduzida/eliminada a informação menos significativa da sample.

```

1  void quant_samples(const std::vector<short>& samples,
2      size_t bits_disc) {
3      for(auto sample : samples){
4          sample = (sample >> bits_disc) << bits_disc;
5          samples_quant.push_back(sample);
6      }

```

Listing 2.1: Código fonte em C

O ficheiro produzido apresenta mais ou menos ruído dependendo do número de bits a descartar.

Como compilar o exercício 3: **make wav_quant**

Comando para executar o programa: **../sndfile-example-bin/wav_quant <input filename> <output filename> <number of bits to discard>**

2.3 Exercício 4

Uma vez que a operação de quantização introduz erros, é importante calcular a quantidade de ruído que introduzimos.

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

Onde E_x é a energia do sinal e tem a seguinte fórmula:

$$\sum_N |x(n)|^2 \quad (2.2)$$

$M(n)$ é o sinal do ruído e tem de equação:

$$x(n) - \bar{x}(n) \quad (2.3)$$

Onde $x(n)$ são samples do áudio original e $\bar{x}(n)$ são as samples de áudio comprimido.

E_r é a energia do sinal do ruído:

$$\sum_N |M(n)|^2 \quad (2.4)$$

É de observar que pelo quanto mais comprimido for o som menor será o valor de SNR.

Assim, para correr o programa o utilizador precisa de fornecer um ficheiro de áudio e de seguida o ficheiro de áudio comprimido correspondente. Se isso se verificar, os cálculos com as fórmulas anteriores vão ser realizados.

```
1  size_t nFrames;
2  double signal_energy = 0, noise_energy = 0;
3  double SNR;
4  double maxError = 0, tmpError;
5
6  while((nFrames = sndFile0.readf(samples_original.data(),
7  FRAMES_BUFFER_SIZE))) {
8      sndFileQ.readf(samples_quant.data(),
9      FRAMES_BUFFER_SIZE);
```

```

9      samples_original.resize(nFrames * sndFile0.channels
10                               ());
11      samples_quant.resize(nFrames * sndFileQ.channels());
12
13      for (long int i = 0; i < (int)samples_original.size
14            (); i++) {
15          signal_energy += pow(samples_original[i], 2);
16          tmpError = abs(samples_original[i] -
17                          samples_quant[i]);
18          noise_energy += pow(tmpError, 2);
19
20          if (tmpError > maxError){
21              maxError = tmpError;
22          }
23      }
24
25      SNR = 10 * log10(signal_energy / noise_energy);
26      cout << "SNR: " << SNR << " dB\nMaximum per sample
27              absolute error: " << maxError << endl;

```

Listing 2.2: Código fonte em C

Como compilar o exercício 4: **make wav_cmp**

Comando para executar o programa: **../sndfile-example-bin/wav_cmp <original file> <quantized file>**

2.4 Exercício 5

No exercício 5, foi nos pedido para produzir efeitos em ficheiros de áudio. Começamos por produzir o eco. Para haver este efeito, temos que pegar em cada sample do som e adicionar uma sample (com k de atraso) que é multiplicada por um ganho ('quantidade' de efeito) que o utilizador pretende efetuar:

$$y(n) = (x(n) + \alpha * x(n-k))/(1+\alpha)$$

Semelhante a este processo, temos o eco múltiplo, para isso já é necessário o programa ir buscar o resultado do eco anterior(função de realimentação) e voltar a processar dando o efeito de eco contínuo.

$$y(n) = (x(n) + \alpha * (y\text{-delay}))/ (1+\alpha)$$

Observámos que, no `single_eco`, como o momento em que isso acontece é único ao longo do áudio torna-se difícil de o efeito ser perceptível.

Além disso, adicionámos o efeito invertido. Para isso, através do ciclo *for* trocou-se a ordem das samples.

```
1  else if(wanted_effect == "reverse") {
2      while((nFrames = sfhIn.readf(samples.data(),
3          FRAMES_BUFFER_SIZE))) {
4          samples.resize(nFrames * sfhIn.channels());
5
6          for (int i = (int)samples.size() - 1; i >= 0; i
7              --) samples_out.insert(samples_out.end(),
8                  samples.at(i));
9      }
10 }
```

Listing 2.3: Código fonte em C

Por último, realizámos uma modulação do som. O som tem duas componentes importantes, a amplitude e a frequência. No caso de uma onda sonora, a amplitude representa um som com uma intensidade mais "alta"ou "baixa".

O fator de modulação vai ser a proporção de variação da amplitude da onda sonora. Como uma onda sonora tem metade do ciclo positivo e metade do ciclo negativo, a fórmula matemática para a modelação de som será:

$$y(n) = x(n) * \cos(2*\pi*(f/fa)*n)$$

```
1  else if(effect=="a"){
2      while((size = sndFileIn.readf(samples_original.data(
3          ), FRAMES_BUFFER_SIZE))) {
4          samples_original.resize(size * sndFileIn.
5              channels());
6          for (int i = 0; i < samples_original.size(); i
7              ++){
8              single_sample_out = samples_original.at(i) *
9                  cos((2 * M_PI * (0.1/fa) * i));
10             samples_out.insert(samples_out.end(),
11                 single_sample_out);
12         }
13     }
14 }
```

Listing 2.4: Código fonte em C

Como compilar o exercício 5: **make wav_effects**

Comando para executar o programa: **../sndfile-example-bin/wav_effects**
<input file> <output_file> <effect>

<effect> é 's' para eco único (single eco), 'm' para eco múltiplo, 'r' para revertido e 'a' para modelação de amplitude. De seguida, mostramos como se executa o programa para cada efeito.

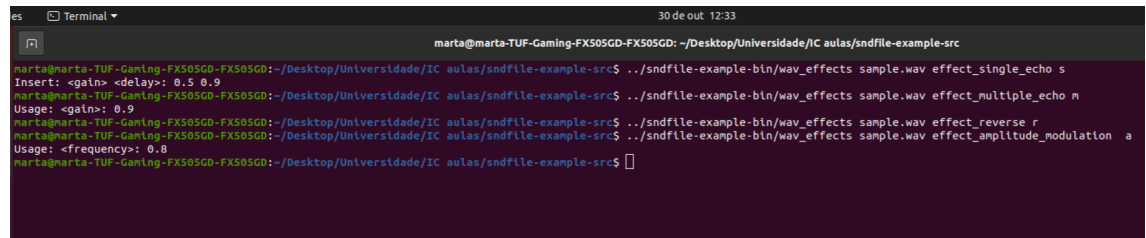
A terminal window with a dark background and light green text. The title bar shows 'es' and 'Terminal'. The top right corner displays the date and time '30 de out 12:33'. The terminal shows the user 'marta' at the prompt 'marta@marta-TUF-Gaming-FX505GD-FX505GD: ~/Desktop/Universidade/IC aulas/sndfile-example-src'. The user enters the command './sndfile-example-bin/wav_effects sample.wav effect_single_echo s'. The prompt changes to 'Insert: <gain> <delay>: 0.5 0.9'. The user enters './sndfile-example-bin/wav_effects sample.wav effect_multiple_echo n'. The prompt changes to 'Usage: <gain>: 0.9'. The user enters './sndfile-example-bin/wav_effects sample.wav effect_reverse r'. The prompt changes to 'Usage: <frequency>: 0.8'. The user enters './sndfile-example-bin/wav_effects sample.wav effect_amplitude_modulation a'. The prompt returns to 'marta@marta-TUF-Gaming-FX505GD-FX505GD: ~/Desktop/Universidade/IC aulas/sndfile-example-src\$'.

Figura 2.4: terminal

Na pasta do nosso projeto encontram-se os ficheiros produzidos nesta compilação.

Capítulo 3

Parte II

3.1 Exercício 6

A classe *BitStream* que nos foi pedida permite a manipulação de um ficheiro bit a bit usando uma stream, sendo o *char* o tipo primitivo mais pequeno possível de representar, ocupando apenas 1 byte de memória.

Para resolvermos como fazer as ações de manipulação de ficheiros criamos uma classe *BitStream*. A assinatura desta classe é demonstrada de seguida:

```
1 class BitStream {
2     private:
3         fstream file;
4         string filename;
5         int mode;
6         int fileSize;
7         std::vector<int> buffer; // stores the bits of the
8                                 // current byte
9         int currentPos; // stores the current bit position
10                        // of reading/writing
11     public:
12         BitStream();
13         BitStream(string filename, char mode);
14         int readBit();
15         std::vector<int> readNbits(int n);
16         void writeBit(int bit);
17         void writeNbits(std::vector<int>);
18         std::vector<int> byteToBuffer(char c);
19         char bufferToByte(std::vector<int>);
20         int getFileSize();
21         void close();
22 };
```

Listing 3.1: Código fonte em C

Como os nomes indicam, temos funções para ler e escrever bit e N bits.

O que se tem de ter em conta na manipulação de ficheiros é que não se pode escrever um único bit diretamente num ficheiro. A unidade de I/O de leitura/escrita é um byte (8-bits). Por isso, tem de se guardar tudo em blocos de 8 bits e depois escrever.

```
1 void BitStream::writeBit(int bit)
2 {
3     if(mode == 1) {
4         cerr << "Cannot write bit in read mode" << endl;
5         exit(1);
6     }
7     // if the current bit position is 8, the byte will be
8     // written and the current position will be set to 0
9     if (currentPos == 8){
10         char byte = bufferToByte(buffer);
11         file.write(&byte, 1);
12         currentPos = 0;
13     }
14     // if the current pos is 0, the buffer needs to be
15     // initialized
16     if (currentPos == 0) buffer = std::vector<int>(8);
17
18     buffer[currentPos] = bit; // put the current bit on the
19     // buffer
20     currentPos++;
21 }
```

Listing 3.2: Código fonte em C

Para a leitura cada carácter que é lido(bit a 0 ou a 1) do ficheiro de entrada será interpretado como um bit do buffer que será utilizado futuramente para a escrita do ficheiro binário.

```
1 int BitStream::readBit() {
2     if (mode == 0) {
3         cerr << "Cannot read file in write mode" << endl;
4         exit(1);
5     }
6
7     if (currentPos == 0) {
8         char byte;
9         file.read(&byte, 1);
10        buffer = byteToBuffer(byte);
11    }
12    int bit = buffer[currentPos];
13    currentPos = (currentPos + 1) % 8;
14    return bit;
15 }
```

15 }

Listing 3.3: Código fonte em C

3.2 Exercício 7

De forma a testar a nossa classe criamos dois programas: *encoder* e *decoder*.

O encoder pega num ficheiro de texto que contém 0's e 1's em que cada conjunto de 8 bits é formado num byte.

Como os valores binários são representados em ASCII e sabendo que '0' em ASCII é 48 e '1' é 49 nós iremos escrever no vetor o valor real lido do ficheiro. Por exemplo, se o valor lido do ficheiro for 1 fica 49-48 que é 1.

```
1 // read data as a block:
2 inputFile.read (buffer,length);
3 cout << "Length of the input file: " << length << endl;
4 cout << "Content of the input file: '" << buffer << "' "
   << endl;
5
6 BitStream BSout (outputFileName, 'w') ;
7
8 //write the bits to the output file
9 vector<int> bits;
10 for (int i = 0; i < length; i++){
11     bits.push_back(buffer[i] - '0'); //
12 }
13 BSout.writeNbits(bits);
14 BSout.close();
```

Listing 3.4: Código fonte em C

Para compilar este encoder basta:

Compilar o encoder: **make encoder**

Comando para executar o programa: **../sndfile-example-bin/encoder <input filename> <output filename>**

Para ver os resultados, basta abrir o output(ficheiro .bin) que foi escrito no terminal.

O decoder faz a transformação oposta.

```
1 //read the bits from the input file
2 vector<int> bits;
3 bits = BSin.readNbits(BSin.getFileSize() * 8);
4 BSin.close();
5
```

```

6      //write the bits to the output file
7      for (int i = 0; i < bits.size(); i++){
8          outputFile << bits[i];
9      }
10     outputFile.close();
11
12     return 0;

```

Listing 3.5: Código fonte em C

Como compilar o decoder: **make decoder**

Comando para executar o programa: **../sndfile-example-bin/decoder <input filename> <output filename>**

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://cplusplus.com/forum/beginner/94731/>

https://www.dir.uniupo.it/pluginfile.php/154365/mod_resource/content/1/Lecture%204_6%20-%20Quantization%20and%20reconstruction.pdf

<https://www.sciencedirect.com/topics/engineering/uniform-quantization>

<https://www.willpirkle.com/forum/algorithm-design/bit-reduction/>