Universidade de Aveiro

Departamento de Eletrónica, Telecomunicações e Informática

Informação e Codificação

Projeto 1

Diogo Fontes (98403), Rafael Amorim (98197), Renato Ourives (98576)

30 de outubro de 2022

Índice

Introdução

De acordo com o solicitado no projeto 1 da unidade curricular de informação e codificação este relatório irá sintetizar todo o raciocínio teórico para a sua finalização.

O código do projeto, tal como, toda a gestão de tarefas encontra-se disponível em:

https://github.com/Raf4morim/IC

Após entrar no link indicado em cima, no diretório ***Lab1*** encontra-se disponível um ***README.md*** com todos os passos a concretizar para gerar e compilar os ficheiros pretendidos.

Foi nos disponibilizada nas aulas uma biblioteca em C, nomeadamente libsndfile, a qual foi utilizada na maioria dos exercícios, permitindo ler e escrever em ficheiros de áudio, este converte automaticamente de um para outro.

Desenvolvimento

Parte 1

Exercício 2

Alterámos o ficheiro “wav_hist” que fornece através um plot, um histograma dos canais de um áudio. Este programa requer como argumentos, o caminho para um ficheiro de entrada, sendo necessário que o ficheiro de entrada seja do tipo .wav e que os canais de saída sejam menores do que os de entrada.

Para a versão mono temos a fórmula que nos dá a média de canais através de um histograma, $(L + R)/2$ e $(L - R)/2$ que correspondem respetivamente ao “Mid Channel” e ao “Side Channel”, para as versões com múltiplos canais.

Para a versão stereo temos dois canais.

Através de vetores registou-se os valores e a contagem de ocorrência de cada canal.

Após realizar os passos do README são gerados os ficheiros ***midChannel.dat*** no caso de o canal ser 0 e ***sideChannel.dat*** no caso de o canal ser 1.

Resultados:

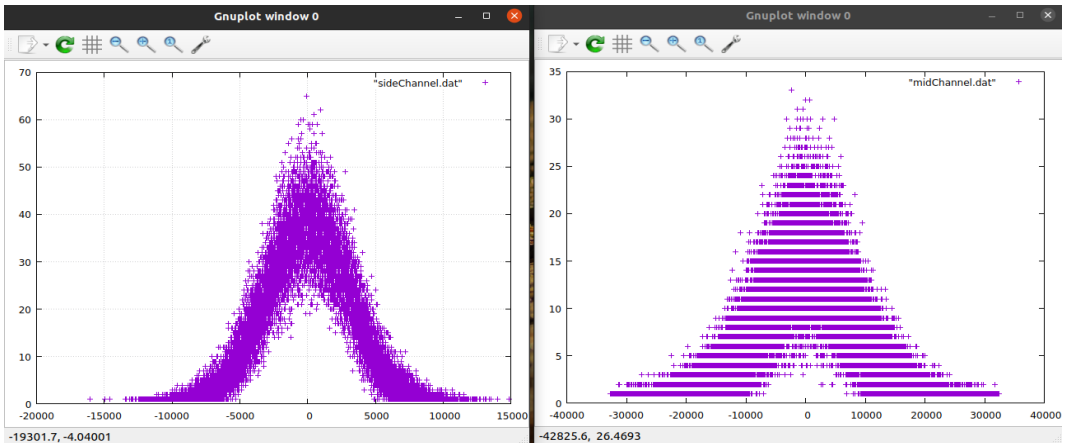


Figura 1: Histograma dos canais

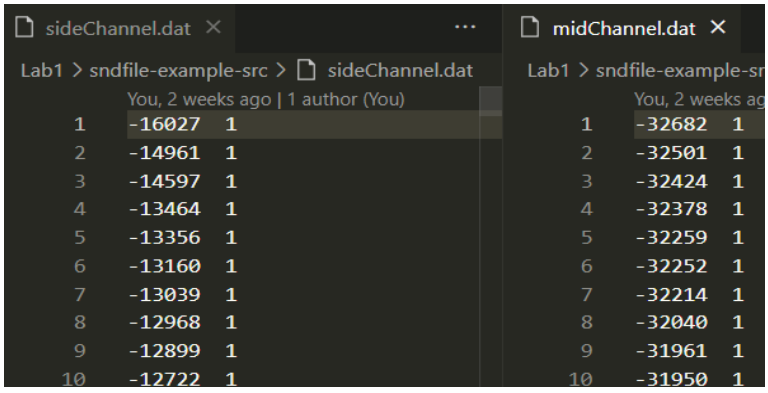


Figura 2: Valores dos canais

Exercício 3

Neste exercício temos como objetivo gerar um ficheiro do formato WAV, que é a réplica de outro, no entanto, devido à redução de bits, basicamente ouve-se um ruído que surge da transformação de sinais contínuos em discretos.

Como argumentos tem de ser inserido um ficheiro de entrada qualificado e de saída do mesmo formato. O último dos argumentos deve ser inserido o número de bits que estão presentes em ambos os ficheiros entre 1 e 15, ou seja, quanto menor mais ruidoso.

Criou-se uma classe ***"wav_quant.h"***, com a função ***"escUn"*** que serve para quantificar de forma uniforme, deslocando os bits desejados para a direita e novamente para a esquerda para "cortar" a sample. Assim, perde-se completamente os bits menos significativos.

A seguir ao clicar no link deve efetuar o download do áudio para ver as diferenças.

smpQuant1bit.wav	…	smpQuant15bit.wav
+ Ruido	…	- Ruido

Exercício 4

Este programa permite determinar com base em cálculos simples a quantidade de ruido entre 2 ficheiros (por exemplo, como input o sample.wav e output smpQuant1bit.wav), dando desta forma continuidade ao problema anterior.

Este cálculo verifica-se através do "Sinal para Ruido" SNR:

$$SNR = 10 \times \log_{10}\left(\frac{Normal}{Distorção}\right)(db), Normal = Energia\ do\ sinal, Distorção = Energia\ do\ Ruido.$$

Decomposição:

$$Normal = \frac{1}{N}\sum_{n=1}^{N} x_n^2; \quad Distorção = \frac{1}{N}\sum_{n=1}^{N}(x_n - \hat{x_n})^2$$

Nestas 2 fórmulas $x_n, \hat{x_n}$ representam, respetivamente, os **samples** do ficheiro original e do quantizado.

Outro valor pedido no exercício era o **erro máximo absoluto por "sample"**. Que tem a seguinte expressão

$$eM = |x_n - \hat{x_n}|$$

Por fim, juntamente com as fórmulas apresentadas foi aplicar de forma sequencial para percorrer as samples todas e daí chegou-se à conclusão que quanto maior o número de bits que se cortava, ou seja, menor o número bits em comum, mais ruído, logo menor SNR.

Resultado:

```
passwd@RafAmorim:/mnt/c/Users/repol/Documents/GitHub/IC/Lab1/sndfile-ex
ample-src$ ../sndfile-example-bin/wav_cmp sample.wav smpQuant1bit.wav
SNR is -8.65457
Maximum per sample absolute error is 1.99997
```

Figura 3: Resultado wav_cmp

Exercício 5

Neste exercício foi-nos imposto implementar um programa chamado ***"wav_effects"***, que produzisse efeitos de áudio, tais como, **"Single Echo"," Multiple Echo"** e **"Amplitude Modulation"**, entre outros.

No Single Echo, pegou-se em cada sample e adicionou-se outra com um atraso multiplicado pelo ganho:

$$\boldsymbol{Y(n)} = \frac{x(n) + \alpha \times x(n-d)}{1+\alpha}, onde\ \alpha = ganho\ e\ d = atraso$$

No Multiple Echo, reutiliza-se o Single Echo:

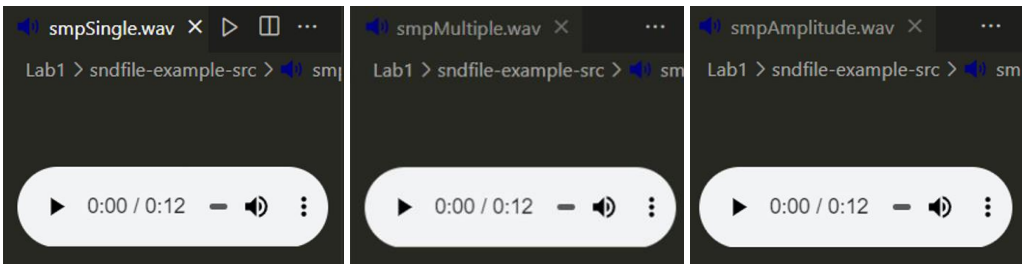
$$Y(n) = \frac{x(n) + \alpha \times (\boldsymbol{y} - delay)}{1+\alpha}, onde\ \alpha = ganho\ e\ d = atraso$$

Na Amplitude Modulation, temos simplesmente de pedir a frequência ao utilizador e aplicar a seguinte expressão:

$$Y(n) = x(n) \times \cos\left(2\pi \times \frac{f}{f_s} \times n\right)$$

No que diz respeito à inserção de argumentos temos de indicar qual o áudio de entrada o nome do ficheiro que queremos produzir e qual o efeito desejado, no caso de ser **"SingleEcho"** surgirá um input a pedir o ganho e o atraso que o utilizador pretender e adicioná-los á "sample" original. **"MultipleEcho"** é necessário introduzir, tal como no **"SingleEcho"**, o ganho e o atraso. **"AmplitudeModulation"** basta apenas indicar a frequência desejada.

Resultados:




```
passwd@RafAmorim:/mnt/c/Users/repol/Documents/GitHub/IC/Lab1/sndfile-example-src$ ../sndfile-e
xample-bin/wav_effects sample.wav smpSingle.wav SingleEcho
SingleEcho
Digite o ganho: 2
Digite o atraso: 40000
```

Figura 4: Resultado do wav_effects

Parte 2

Exercício 6

Desenvolve-se uma classe nomeada por "**BitStream.hpp**", com o objetivo de manipular ficheiros, lendo e escrevendo bits, de um para o outro. Sendo posteriormente utilizado no "**encoder.cpp**" e no "**decoder.cpp**".

Para esta classe, tivemos de ter em consideração que não se pode escrever apenas 1 bit diretamente de um ficheiro para o outro. Este precisa de passar por dentro do buffer e só quando este estiver cheio é que escreve no ficheiro byte a byte.

Daí a termos usado uma estrutura de dados que representa um buffer, pois este será percorrido sequencialmente e serve para a conversão entre bits e bytes,

Byte → Bit na operação de **leitura** e **Bit → Byte** na operação de **escrita**.

Exercício 7

Tal como foi já referido anteriormente, o exercício anterior necessita de ser testado e para tal implementou-se o **"encoder.cpp"** e **"decoder.cpp"** que servirá de teste.

Ao correr o **"encoder.cpp"** com um ficheiro de entrada ".txt" com '0's e '1's vai gerar um ficheiro com texto formal, ao qual chamou-se **"encExemplo.txt"** depois vê-se o inverso no **"decoder.cpp"** gerando o "decExemplo.txt" que se resume em retornar o mesmo que tinha no ficheiro inicial, tal como vemos na imagem de lado.

Nas imagens a baixo vemos como as operações são opostas.

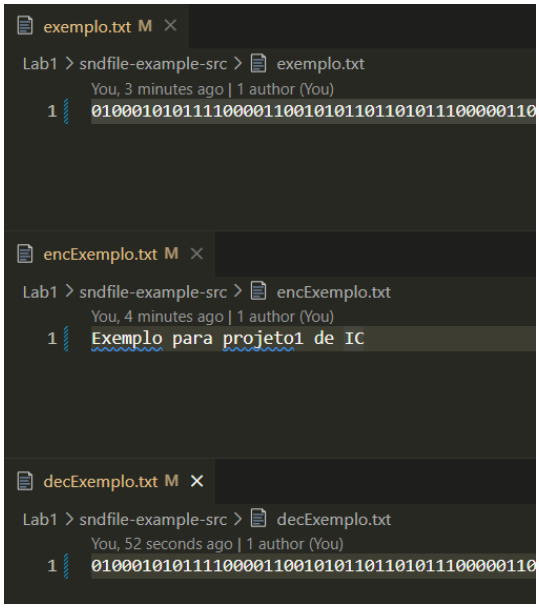


Figura 5: Ficheiro gerados

```
vector<int> bits;
for (long unsigned int i = 0; i < buf.length(); i++){
    bits.push_back(buf[i] - '0');
}
oF.writeBits(bits);
```

```
for (long unsigned int i = 0; i < bits.size(); i++) outputFile << bits[i];
```

Figura 6: Excertos de código

Parte3

Exercício 8

Não conseguimos chegar à solução final deste exercício.

Contribuição dos autores

- Diogo Fontes – 46.5 %
- Rafael Amorim – 46.5 %
- Renato Ourives – 7%

Bibliografia

[1] Armando J. Pinho. Some Notes For the Course Information and Coding,2022