ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Trabalho 1
Comunicações e Processamento de Sinais

LICENCIATURA EM ENGENHARIA INFORMÁTICA E MULTIMÉDIA
ANO LETIVO 2017/2018

Turma LEIM 31D
Grupo:13
Data:25/11/2018
Docentes:
Eng. André Lourenço
Eng. Pedro Fazenda
Alunos:
Luis Fonseca (A45125)
Philipp Al-Badavi (A45138)

Índice

Introdução

O 2º trabalho da disciplina Comunicação e Processamento de Sinais consistia na continuação da implementação de blocos do esquema de envio e receção de informação usando a modulação digital. Desta vez foi nos solicitado a implementação de blocos como , bloco de codificação , bloco de controlo de erros para o tal foi utilizado o Hamming(7,4), bloco de detecção e correcção de erros e por fim o bloco de descodificação. Isto tudo é necessário para a verificar se a mensagem recebida tem erro e se possível a correção do mesmo.

1. Este exercicio consistia na criação do bloco de codificação e do bloco de descodificação do sinal.No bloco de codificação, para o tal foi utilizada a função “binary_repr” da biblioteca Numpy, o grupo optou pela utilização dessa função pois com ela havia a possibilidade de converter um número decimal num número inteiro com quantidade de bits que a pessoa quissesse. Após a conversão o grupo deparou que o tipo do array criado depois da utilização desta função era string e como os Engenheiros aconcelharam com que a função devolvesse o array onde cada número inteiro fosse ocupar uma posicação no array, foram feitos dois “joins” e um split para separar o número um a um e por final no retorno o array devolvido foi transformado num inteiro com a função “astype()”. De seguida, a outra função que ainda pretence ao bloco de codificação é a função “arrayDeCod”, está função consistia num simples resize da lista fornecida,conforme o tamanho fornecido.

```
intTObin =
[0 0 0 1 1 0 1 1]
arrayDeCod =
[[0 0]
 [0 1]
 [1 0]
 [1 1]]
```

Figura 1 - Ex. Converção de um array decimal de 0 a 3(inclusive) para um binário com 2 bits

Por final foi implementada a função de descodificação que pegava num array binário convertia o para um array do tipo string e transformava os números binários em números decimais, retornando um array com os valores decimais do tipo int.

```
array inicial
[0 1 2 3]
intTObin =
[0 0 0 1 1 0 1 1]
arrayDeCod =
[[0 0]
 [0 1]
 [1 0]
 [1 1]]
binTOint =
[0 1 2 3]
```

Figura 2- Exemplo de funcionalidade deste exercício

2. Neste exercício foi nos solicitado a utilização das funções já criadas para fazer a codificação e descodificação do sinal, a medição do SNR e como ponto final ouvir o sinal descodificado. Depois de ouvir o som final foi possível ver que a disturção é tão grande no sinal que nem se percebe nada , mas nota se que quanto maior for a quantidade de bits menos disturção irá haver.

3. Este exercício consistia na criação do bloco de controlo de erro, para o tal foi implementado em python a função de Hamming(7,4) que para cada 4 bits gerava mais 3 que eram o resultado do código de Hamming, transformando uma mensagem de 4 bits numa de 7bits, em Python isto é feito com a função do Numpy "dot" que multiplica a matriz geradora pelo array fornecido. O resultado obtido é essencial na medida em que é utilizado no calculo do Sindroma para saber qual é o bit que esta errado. Sabendo que o Hamming precisa inicialmente de uma mensagem de 4 bits, também foi implementado uma resolução que caso a mensagem não tiver os 4 bits, irá haver adição de bits ,zeros, necessários para que a mensagem tenha a quantidade de bits necessários. No final, com a função do Numpy "hstack" é adicionado no início do array que será retornado, a quantidade de bits adicionados. Esse número é depois utilizado no outro exercício para retirar os bits adicionados.

```
Sinal antes do hamming
[0 0 0 1 0 1 0 0]
Sinal depois do hamming
[0 0 0 0 0 1 1 1 1 0 1 0 0 1 1 0]
```

Figura 4 - Resultado depois de passar o sinal pelo hamming

4. Neste exercício foi nos solicitado a criação do bloco que corrige os erros da mensagem. Para o tal foram criadas 4 funções,a função “retiraBitsAdicionados”, que consiste separar os primeiros 2 indices do array de mensagem, a função “mensagem” é um resize da mensagem ,transforma a mensagem numa matrix de 7 por X(depende do tamanho da mensagem), a função Sindroma pega na matriz geradora e na matriz de mensagem e através da função do Numpy “dot” retorna o sindroma da mensagem, esse sindroma irá indica o indice da mensagem que apresenta um erro, por final a função de “correccaoDoErro” pega na mensagem, através do sindroma vê o índice que apresenta erro, corrige o erro , tira os 3 bits resultantes do codigo de hamming através de uma “list comprehension” , retira os bits que foram adicionados na função de Hamming(7,4) e devolve a mensgem corrigida.

```
array inicial
[0 0 1 0 1 0]
array depois de hamming(7,4)
[0 0 1 1 1 0 1 1 0 0 0 0 1 1]
array depois de retirarem os bits a mais e depois de passar pela mensagem
[[0 0 1 1 1 0 1]
 [1 0 0 0 0 1 1]]
Sindroma do array
[[1 1 1]
 [0 0 0]]
array corrigido
1
[0 0 1 0 1 0]
```

Figura 5 -Exemplo de funcionalidade das funções

5. Neste exercício foi nos solicitado o calculo do SNR de mensagem inicial e final , o calculo do BER antes da correcção de depois da correcção e a sua comparação. O grupo fez o calculo de SNR .De seguida o grupo calculou o BER e BER' e deparou que os seu valores diferiam um do outro, tanto no SNR como nos BERs, isto tudo pois um erro era aplicado a função, erro este foi gerido por uma função fornecida o "y = 1 ∗np.logical_xor(x, np.random.binomial(1, BERt, len(x)))" , onde o "x" era o nosso sinal codificado e mesmo depois de correcção , como o código de Hamming(7,4) só é capaz de corrigir um erro , continuavam a haver erros no final o que explica o porque dos valores serem diferentes .

```
valor do BER1:
0.3571428571428571
valor do BERlinha1:
0.3333333333333333
----------
valor do BER2:
0.21428571428571427
valor do BERlinha2:
0.16666666666666666
----------
valor do BER3:
0.6428571428571427
valor do BERlinha3:
0.5

SNR a 3bits
[24.84968223 36.88968223 54.94968223]
[16.75099427 28.79099427 46.85099427]

SNR a 5bits
[16.86115744 28.90115744 46.96115744]
[16.86115744 28.90115744 46.96115744]

SNR a 8bits
[16.94178645 28.98178645 47.04178645]
[16.94178645 28.98178645 47.04178645]
```

Figura 6 - Resultado de calculo de BERs e SNRs

Conclusões

No final do trabalho os objetivos foram quase todos alcançados, foi possível fazer a codificação e a descodificação do sinal como também a correcção do erro. Porém o grupo não conseguiu reproduzir de maneira correta os sons descodificados, pois o som reproduzido era diferente do inicial, mas foi possível destacar que quanto maior foir o número de bits na quantização menor sería a dísturção do som reproduzido.

Bibliografía

Sebenta 1 - Carlos Eduardo De Meneses Ribeiro
III-1 Códigos detectores e correctors – André Lourenço