

1º Semestre 16/17 SI

Comunicação e Processamento de Sinais

**Relatório do 3º Trabalho Prático**

Eng. Pedro Fazenda

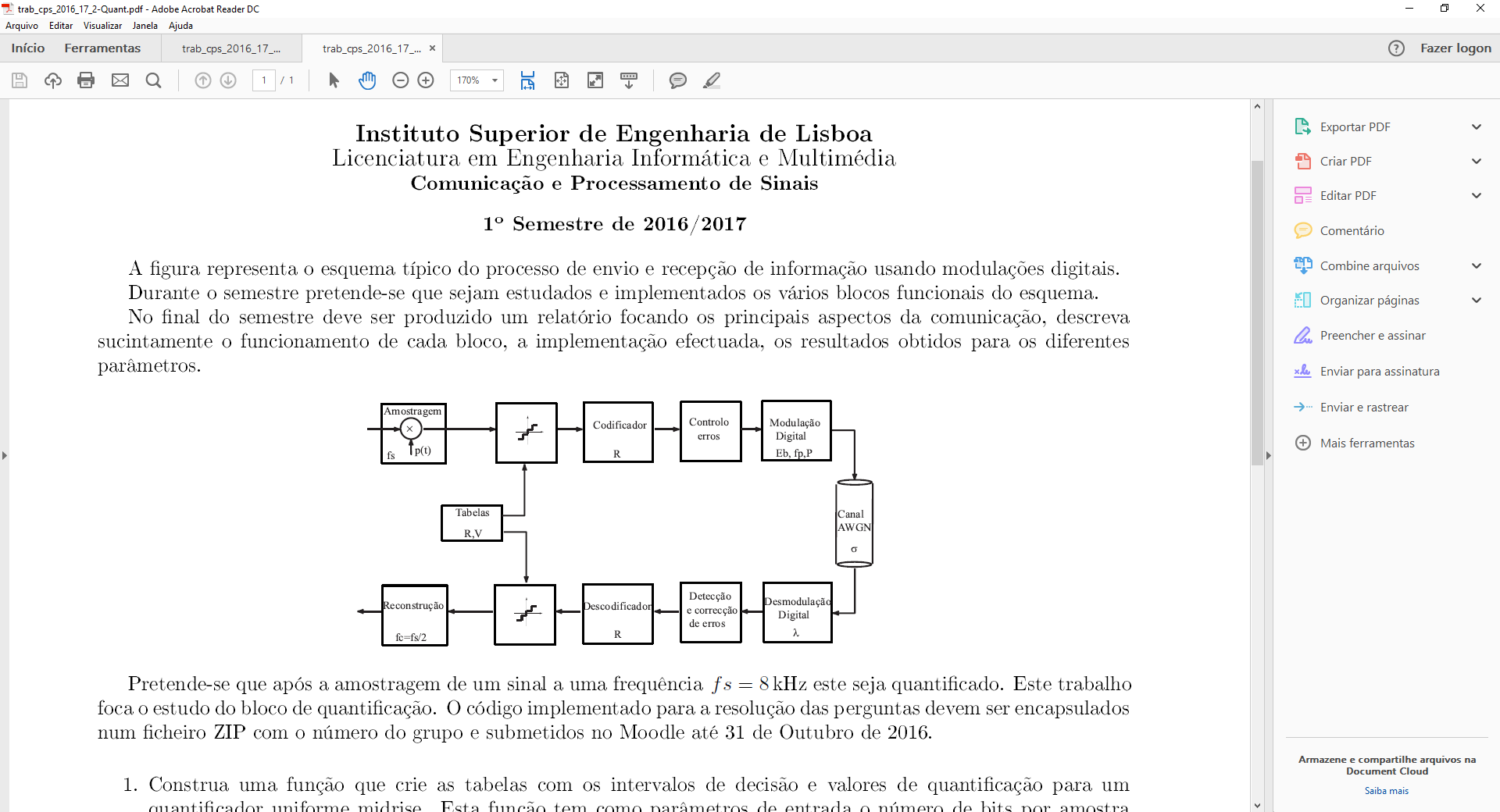
27/11/2016

**Trabalho realizado por:**

Diogo Fernandes nº39205

Rui Santos nº39286

**Introdução**



Este terceiro trabalho foca-se nos blocos de Controlo e Correção de erros e suas operações inversas, iremos utilizar código desenvolvido no primeiro e segundo trabalho para executar testes que pretendemos neste trabalho.

Exercícios

1. Construa uma função que dado um array de N bits, para cada 4 bits gere 7 bits, sendo 3 bits resultado do código de Hamming(7,4).

def hamming(message):  
 currentMessage = message  
 #verificacao do tamanho da mensagem  
 resto = len(currentMessage)%messageBits  
 #extenca do array inicializada  
 if(not (resto == 0)):  
 finalSize = len(currentMessage)+(messageBits-resto)  
 currentMessage = np.zeros(finalSize)  
 #preenchimento binario correto  
 for i in range(len(message)):  
 currentMessage[i] = message[i]  
 #matriz identidade  
 identity = np.eye(messageBits)  
 #matriz geradora  
 G = np.hstack((identity,P))  
 #array de bits final  
 size = len(currentMessage)+((len(currentMessage)/messageBits)\*controlBits)  
 finalBits = np.zeros(int(size))  
 count = 0  
 for i in range(0,len(currentMessage),messageBits):  
 currentSlice = currentMessage[i:i+messageBits]  
 #bits de paridade resultantes  
 addControl = np.mod(np.dot(currentSlice,G),2)  
 for x in range(0,len(addControl)):  
 finalBits[count] = addControl[x]  
 count += 1

return finalBits.astype('int16')

**2.**Construa uma função que dado um array de M bits para cada 7 bits detete e corrige possiveis erros, sendo o resultado 4 bits de mensagem.

def sindrome(message):  
 #possiveis sindromas  
 sindromeTable = np.array([[0,1,1],[1,1,0],[1,0,1],[1,1,1],[1,0,0],[0,1,0],[0,0,1]])  
 #percorre o sinal  
 for i in range(0, len(message), totalBits):  
 currentMessage = message[i:i+totalBits]  
 #resultado do sindroma multiplicado por matriz de teste de paridade  
 sindromeResult = np.mod(np.dot(currentMessage,sindromeTable),2)  
 size = (len(message)/totalBits)\*messageBits  
 correctedMessage = np.zeros(int(size))  
 index = 0  
 count = 0  
 errors = 0  
 for x in range(0,len(sindromeTable)):  
 if(sum(sindromeTable[x]==sindromeResult)==controlBits):  
 message[i+count] = (message[i+count]+1)%2  
 #apenas corrige um erro  
 break  
 else:  
 count+=1  
 if(sum(sindromeResult)!=0):  
 errors += 1  
 count =0  
 correctedMessage[index:index+messageBits] = message[i:i+messageBits]  
 index += messageBits

return errors, correctedMessage.astype('int16')

**3.**Com as funções anteriores e admitindo que os restantes blocos do emissor, canal e receptor possam ser simulados pela expressão:

Onde BERt, contém o valor pretendido para o bit error rate. Meça a SNR na recepção, o BER antes e após a correção de erros, para diferentes valores de BERt.

def simulation():  
 fsRecord, data = wav.read(caminho+"sinaldevoz8khz.wav")  
 R = 8  
 BERt = 0.1  
 #criacao das tabelas  
 VD,NQ = createTable(R,np.max(np.abs(data)))  
 #quantificacao do sinal lido  
 SQ, IQ = quantificacao(data,NQ,VD)  
 #codificacao do sinal quantificado  
 signalCodif = codificaSinal(IQ,R)  
 #adicao dos bits de erro  
 signalControl = hamming(signalCodif)  
 #aplicacao de ruido  
 noise = 1\*np.logical\_xor(signalControl, np.random.binomial(1,BERt, len(signalControl)))  
 #correcao do sinal  
 correctedB, corrected = sindrome(noise)  
 #medicao do BER sem correção  
 BerScorrecao = sum(np.logical\_xor(signalControl,noise))/float(len(signalControl))  
 print ("BER sem correcao")  
 print (str(BerScorrecao))  
 print("\n")

#medicao do BER apos correção  
 #comparando o array inicialmente codificado com o resultante  
 #da correção  
 erro=0  
 for i in range(len(signalCodif)):  
 if(signalCodif[i] != corrected[i]):  
 erro = erro +1

berCcorrecao = float(erro)/float(len(signalCodif))  
 print ("BER com correcao")  
 print (str(berCcorrecao))  
 print("\n")

Foi criada uma função que simulasse todo o processo desde a leitura de um sinal, quantificação do mesmo, codificação, adição dos bits de controlo, aplicação de ruído e todo o processo inverso.

#descodificacao do sinal   
decodedSignal = descodificaSinal(corrected,R) #desquantificacao do sinal  
signalQuant = quantificacaoInversa(decodedSignal,NQ)  
## medicao da SNRPratica  
erro = erroQuantificacao(data,signalQuant)  
potenciaErro = potenciaErroQuant(erro)  
potencia = potenciaSinal(data)  
SNRP = SNRPratico(potencia,potenciaErro)  
print("SNR Pratica do sinal")  
 print(SNRP)  
 print("\n")

#gravacao do sinal apos todo o processo recordSignal("sinalteste8kR8.wav",fsRecord,signalQuant.astype('int16'))

**4.**Admita o bloco com a mensagem [1,0,1,0]

**a)** Através do seguinte polinómio gerador, , gere a mensagem com os bits de controlo de erros.  
  
Conhecendo o polinómio, concluímos que a palavra correspondente é [1,0,1,1]  
Sabemos também à partida que é um Hamming (7,4) logo temos 3 bits redundantes.  
Sendo possível então obter a mensagem com os bits de controlo da seguinte forma:

k = 7  
n = 4  
q = 3

Aplicando a divisão pelo polinómio, sendo o resultante o bits de controlo finais para adicionar à mensagem.  
  
Mensagem final a transmitir: 1010 011

1010000 | 1011  
1011 1001  
00010  
 0000  
 00100  
 0000  
 01000  
 1011  
 0011 - Resto

**b)** Admita que recebe a mensagem [1,1,0,1,1,0,1]. O que pode concluir?

Aplicando a mesma divisão da alínea anterior:

Concluimos que existe erro no bit 4, consultando a tabela de síndromas do hamming (7,4)

1101101 | 1011  
1011 1111  
01101  
 1011  
 01100  
 1011  
 01111  
 1011  
 0100 - Resto

**c)**Admitindo que o sistema de transmissão funciona a 10Mbits/s com um BER inicial de 10-3, qual o BER após a correção do controle de erros?

Consultando o formulário, usamos a fórmula:

Concluímos que o BER apos a correção diminuiu, havendo assim menos bits errados.

**d)** Nas condições das alínea anterior, qual o tempo médio entre dois bits errados(com e sem deteção de erros)?

- 10Mbits/s = 10 000 000 bit/s  
  
- Tempo médio de bit sem correção:

- Tempo médio de bit com correção:

Para calcularmos, temos de saber o ritmo binário na receção após a correção de erros, através da relação em razão de código, conseguimos calcular o valor pretendido.

bit/s

Concluímos que, se o tempo entre erro de bit aumenta, indica-nos claramente que existem menos erros no sinal.