ISEL | DEETC

Comunicações

2° Trabalho Práctico

Semestre de Inverno de Ano 2009/2010

Autores:
30896 – Ricardo Canto
31401 – Nuno Cancelo
33595 – Nuno Sousa

Indície

Enunciado

Instituto Superior de Engenharia de Lisboa
Licenciatura em Engenharia Informática e de Computadores
Comunicações

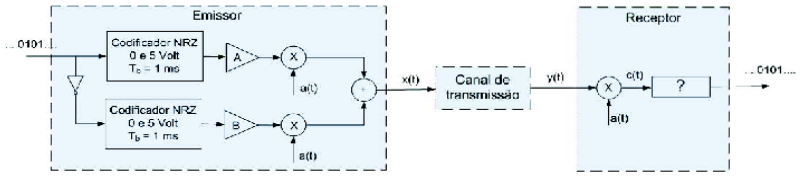
Segundo Trabalho Prático
Semestre de Inverno 2009/2010 (4 de Dezembro de 2009)
Data limite de entrega do exercício 1: 8 de Janeiro de 2010
Data limite de entrega dos exercícios 2, 3 e relatório completo: 12 de Fevereiro de 2010

Objectivos:

- Desenvolvimento de programas em MATLAB.
- Estudo de técnicas de comunicação digital. Análise de desempenho na presença de ruído.
- Estudo de técnicas de detecção e correcção de erros.

O código deverá ser entregue em formato electrónico e o relatório em papel.

1. A figura ilustra o sistema de comunicação digital (SCD) desenvolvido no primeiro trabalho prático.



   a) Recorrendo à função awgn do MATLAB, escreva a função tx1 que realize um canal de transmissão, tal que este some ruído branco e gaussiano ao sinal de entrada, com relação sinal-ruído especificada como parâmetro.

   b) Modifique a função anterior de forma a que, para além da adição de ruído, o canal de transmissão realize também filtragem do tipo passa-baixo com frequência de corte programável (função tx2).

   c) Realize um único receptor que sirva para as modulações PSK e OOK. A escolha da modulação é indicada por um parâmetro.

   d) Avalie o BER (Bit Error Rate) do sistema para as modulações PSK e OOK usando os canais de transmissão tx1 e tx2, para mensagens de teste à sua escolha. Comente os resultados obtidos.

---

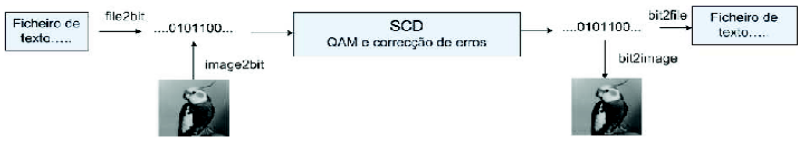
2. Considere o SCD desenvolvido, bem como os canais de transmissão considerados no exercício anterior.

   a) Trace as curvas experimentais de BER para as modulações OOK e PSK, usando o canal de transmissão tx1. Compare e comente as curvas obtidas.

   b) Adicione ao SCD um código corrector de erros, com t=2 bits em erro por cada bloco de n bits. Apresente exemplos que mostrem o funcionamento do mecanismo de correcção.

   c) Compare as curvas de BER da modulação PSK, sem e com o código corrector de erros.

3. Realize o menor número de modificações sobre o SCD implementado de forma a que este opere com modulação QAM, com M níveis à sua escolha. O sistema deverá utilizar o código corrector de erros do exercício anterior. Recorrendo às funções file2bit, bit2file, image2bit e bit2image, utilize o sistema para transmissão de ficheiros de texto e imagens. Avalie o BER para diferentes valores de relação sinal-ruído.



Exercício 1

O desenvolvimento deste grupo foi simples de executar, exceptuando uma ou outra dúvida na implementação.

Alínea A

A realização desta (e da próxima) alínea foi tradução directa do que é solicitado no enunciado.

```
function [Y]=tx1(signal,SNR)
    Y=awgn(signal,SNR);
end
```

Alínea B

Aproveitamos a implementação anterior para realizarmos esta nova alínea. Uma vez que era solicitado uma filtragem, implementamos igualmente uma função para tal proposito e que poderia ser reutilizada por mais funções.

```
function [Y]=tx2(signal,SNR,FS,Fc)
    signal_F=fftshift(fft(signal));
    Filtro=filtroPassaBaixo(length(signal_F),Fc);
    signal=ifft(signal_F.*Filtro);
      Y=tx1(signal,SNR);
end

function [Filtro] = filtroPassaBaixo(fs,fc)

    Filtro=1:fc/fs:fc;
    rightLimit=floor(fc/2);
    for i=1:length(Filtro)
        if (i>rightLimit )
            Filtro(i)=0;
        else
            Filtro(i)=1;
        end
    end
end
```

Assim desta forma obtemos o nosso canal de transmissão que recebe o sinal pretendido, o nível de SNR, o tipo de canal a utilizar e a respectiva frequência de corte (a ser utilizada no canal tx2).

```
function [Y]=TX(signal,SNR,type,FS,Fc)
    switch type
     case 1
        Y=tx1(signal,SNR);
      case 2
        Y=tx2(signal,SNR,FS,Fc);
    end
end
```

Alínea C

Na implementação deste módulo temos em consideração o foi assimilado nas aulas durante o semestre de forma a que pudéssemos recuperar o sinal original.

Assim, o nosso receptor terá um conjunto de módulos para procederemos em conformidade com o principio teórico de um receptor.

```
function [signal] = receptor(signal,FS,TB,modulationType)
    signal=correlador(signal,FS,TB,modulationType);
end


function [result]= correlador(Signal,FS,Modulation)
    [TB,Amp]=NRZValues();
    %construção do array de tempos
    bitsLen = length(0:1/(FS-1):TB);
    bitsnbr = length(Signal)/bitsLen;
    t = 0:(TB*bitsnbr)/(bitsLen*bitsnbr-1):TB*bitsnbr;

    Signal=Signal.*cos(2*pi*10000*TB);
    tmpY=zeros(1,bitsLen);
    result=zeros(1,bitsnbr);

    for i=1:bitsnbr
        count=1;
        while (count <= bitsLen )
            idx=(i-1)*bitsLen+count;
            tmpY(count)=Signal(idx);
            count=count+1;
        end
        Y=sum(tmpY);
        result(i)=decisor(Y,Modulation);
    end
end
```

```
function [result] = decisor(Signal,Modulation)
    [TB,Amp]=NRZValues();
    if (strcmpi(Modulation,'OOK'))
            Eb=Amp*Amp*TB/2;
             if (Signal > Eb/2)
                 result = 1;
             else
                 result =0;
             end
    elseif(strcmpi(Modulation,'PSK'))
                if (Signal > 0)
                 result = 1;
             else
                 result = 0;
             end
    else
        fprintf('Modulação indicada é inválida.\n');
        result=-1;
        return;
    end
end
```

Alinea D

Para realizarmos o solicitado nesta alínea, construímos mais um módulo que nos permite obter o BER dados os sinais de entrada e os sinais de saída.

```
function Result=BER(In,Out)
    Result=sum(abs(In ~= Out));
    Result=Result/length(In);
end
```

<introduzir resultados>

Exercício 2

Para este exercício procedemos a um ciclo para guardar os valores obtidos através da função anterior para cada tipo de sinal de forma a podermos esboçar o gráfico.

Alínea A

Dados os resultados (incorrectos) da alínea anterior não nos é possível esboçar os gráficos de um forma conclusiva.

Alínea B

Para implementarmos um código corrector t=2 erros, teremos que ter uma palavra com distância mínima de Hamming igual ou superior a 2*t+1, ou seja, com uma distância superior a 5.

Continuando a análise podemos inferir que $5 \leq 1+n-k$, assim sendo o nosso código(N,K) terá que no mínimo diferir em 4 unidade entre os valores.

Após a construção da tabela conseguimos obtemos uma matriz para um código(11,4) com distância mínima de Hamming de 5, que nos permitirá detectar até 3 bits de erro e corrigir até 2 erros.

Uma vez que o código corrige até 2 bits em erro se existirem 4 bits em erro serão corrigidos para uma palavra de código errada como se existissem 2 bits em erro de outro Sindroma.

M0	M1	M2	M3	M0+M1+M2	M0+M1+M3	M1+M2+M3	M0+M2+M3	M1+M3	M1+M2	M0+M3	dH
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	1	0	1	0	0	1	5
0	1	0	0	1	1	1	0	1	1	0	6
1	1	0	0	0	0	1	1	1	1	1	7
0	0	1	0	1	0	1	1	0	1	0	5
1	0	1	0	0	1	1	0	0	1	1	6
0	1	1	0	0	1	0	1	1	0	0	5
1	1	1	0	1	0	0	0	1	0	1	6
0	0	0	1	0	1	1	1	1	0	1	6
1	0	0	1	1	0	1	0	1	0	0	5
0	1	0	1	1	0	0	1	0	1	1	6
1	1	0	1	0	1	0	0	0	1	0	5
0	0	1	1	1	1	0	0	1	1	1	7
1	0	1	1	0	0	0	1	1	1	0	6
0	1	1	1	0	0	1	0	0	0	1	5
1	1	1	1	1	1	1	1	0	0	0	8

Uma vez construída a matriz procedemos ao desenvolvimento do módulos de geração, detecção e correcção de erros.

Antes de gerar o código criámos uma função que normaliza o tamanho da mensagem a codificar para que esta fique com um múltiplo de 4 elementos. Deste modo evitamos erros de dimensão de matrizes no MatLab.

O módulo para gerar o código simplesmente multiplica a mensagem pela matriz geradora construída a partir da tabela acima.

```
function [sinal_out] = geraCodigo(sinal)
%Função que recebe um sinal digital codifica a mesma segundo um código
11,4
%e devolve o sinal já codificado
```

```
%nº de bits de mensagem
numBits=4;

i=1;
k=0;
aux =[0,0,0,0];
sinal_out=[];
while (i*k < length(sinal))
    for i=1:numBits
        aux(i)=sinal(numBits*k+i);
    end
    sinal_out = horzcat(sinal_out,codificaBloco(aux));
    k = k+1;
end
end


function [codigo] = codificaBloco(mensagem)
%Função que recebe o código de 4 bits a transmitir e devolve o código de
11
%bits já codificado
%         [M0,M1,M2,M3,    B1,B2,B3,B4,B5,B6,B7]
geradora=[[1 0 0 0          1 1 0 1 0 0 1]
          [0 1 0 0          1 1 1 0 1 1 0]
          [0 0 1 0          1 0 1 1 0 1 0]
          [0 0 0 1          0 1 1 1 1 0 1]];
codigo = mod(mensagem * geradora,2);
end
```

No módulo de descodificação do código começámos por criar a matriz geradora de paridade P através da parte direita da matriz geradora. Desta é construída a matriz de Controlo de Paridade H. Esta matriz associada a uma matriz identidade 11x11 dá-nos os Síndromas de erro de 1 bit (11 possibilidades).

Ao multiplicar a matriz H por uma matriz "identidade" em que as linhas têm todas as combinações de 11, 2 a 2 obtemos a matriz com todos os Síndromas de erro de 2 bits (55 possibilidades).

Multiplicando o sinal recebido pela matriz H^T obtemos o síndroma associado a esse sinal. Passando pela matriz com todos os Síndromas de erro construída anteriormente (incluindo a ausência de erro – síndroma todo a 0) obtemos a linha correspondente na matriz "identidade" referida acima que, somada ao sinal recebido resulta no mesmo com os bits corrigidos.

```
function [sinal_out] = descodificador(sinal)

%n° de bits de mensagem
numBits=4;

%Sub-matriz Geradora Paridade P
P=[[1,1,0,1,0,0,1]
   [1,1,1,0,1,1,0]
   [1,0,1,1,0,1,0]
   [0,1,1,1,1,0,1]];

%Matriz Controlo Paridade H
%H' é a matriz com os sindromas de erro de 1 bit (11 possibilidades)
H = horzcat(P',eye(7));

%Agora é necessário construir a matriz dos sindromas para erros de 2 bit
%Sera uma matriz com 55 sindromas (Combinações de 11, 2 a 2)
    k=1;
    for i=1:10
        for j=i+1:11
            m_sindromas(k,i)=1;
            m_sindromas(k,j)=1;
            k=k+1;
        end;
    end;

%Matriz com os bits a corrigir para cada sindroma
corrigeBits = vertcat(zeros(1,11),eye(11),m_sindromas);
%corrigeBits
m_sindromas=0;
%Matriz com todos os sindromas de 2 bits
%calculada multiplicando a matrix de correcção com a Matriz Controlo
Paridade transposta
    for i=1:length(corrigeBits)
        m_sindromas(i,1:7)= mod((corrigeBits(i,1:11)*H'),2);
    end;
    %m_sindromas

%Processamento do sinal de entrada
%Por cada bloco de 11 bits, verifica se existe erro e se sim, corrige o
%mesmo construindo o sinal final usando apenas os 4 bits de mensagem

sinal_out=zeros(1,length(sinal)/11*4);
    for x=1:11:length(sinal)
        %variavel que permite saber se foi detectado erro fora dos limites
        %de correcção - mais do que dois bits em erro
        sinal_p=0;
        sindroma = mod(sinal(1,x:x+10)*H',2);
        for i=1:length(m_sindromas)
            if (sindroma == m_sindromas(i,1:7))
                    sinal_out(1,(floor(x/11)*4+1):(floor(x/11)*4+1)+3)  =
(mod(sinal(1,x:x+3) + corrigeBits(i,1:4),2));
                sinal_p = 1;
                break;
            end;
        end;
        if(sinal_p==0)
            %se sinal tem mais do que 3 bits em erro então mantem sinal da
            %entrada na saída e informa utilizador
            sinal_out(1,((x-1)/11*4+1):((x-1)/11*4+1)+3) = sinal(1,x:x+3);
%             fprintf('!!!Erro de transmissão!!!\n');
```

```
%               fprintf('Mais de dois bits trocados entre o bit %d e o bit
%d.\n',x, x+10);
%              fprintf('Transmissão deverá ser repetida.');
        end;
    end;
%sinal_out
end
```

Alínea C

Compare as curvas de BER da modulação PSK, sem e com o código corrector de erros.

Exercício 3

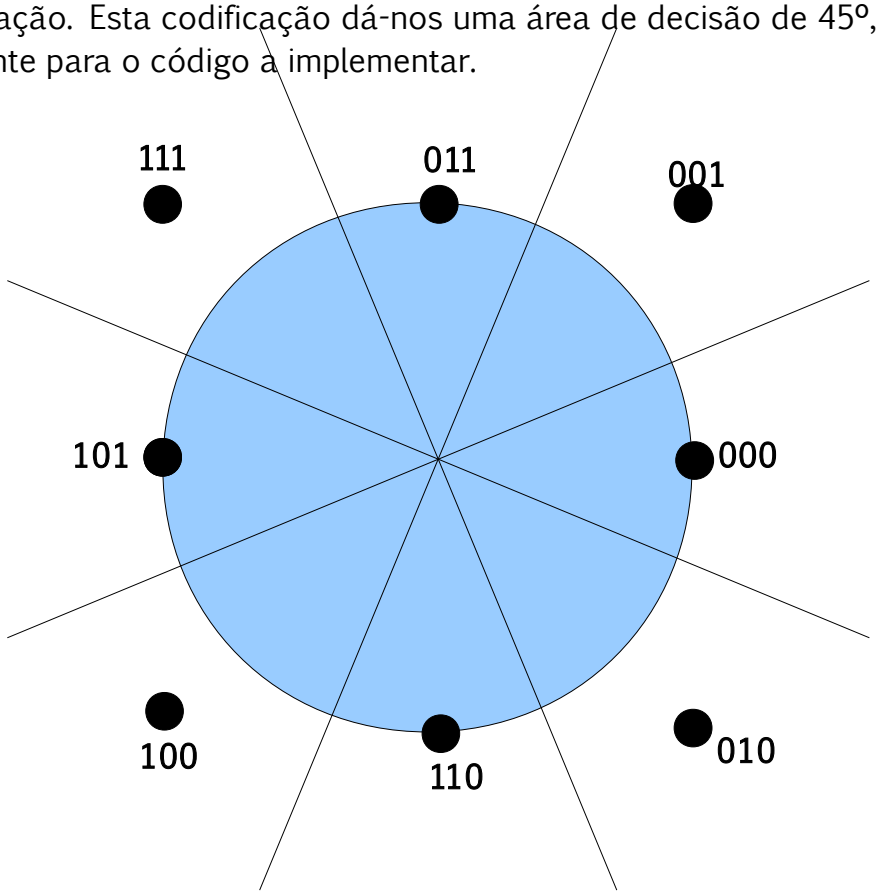
Para procedermos ao solicitado, teremos que realizar alguma alterações ao nosso emissor e receptor.

No nosso emissor modificaremos o nosso módulo modulador para suportar os sinais coseno e seno de forma que possa "emitir" os símbolos referentes aos bits de entrada.

No nosso receptor modificaremos o nosso módulo decisor para suportar a interpretação dos símbolos recebidos.

Tanto no emissor como no emissor temos que codificar/descodificar os símbolos que emitimos/recebemos para poder os interpretar de forma a obtermos o sinal inicial no fim.

Optamos por implementar um código QAM8, que nos dá símbolos de 3 bits de codificação. Esta codificação dá-nos uma área de decisão de 45º, que nos parece suficiente para o código a implementar.



Assim podemos obter a seguinte tabela de codificação/descodificação de símbolos:

Fase		
Limite Mínimo	**Limite Máximo**	**Símbolo**
22,5	67,5	001
67,5	112,5	011
112,5	157,5	111
157,5	202,5	101
202,5	247,5	100
247,5	292,5	110
292,5	337,5	010
337,5	22,5	000

No que diz respeito a alterações especificas que teremos que fazer no emissor serão as seguintes:

O nosso modulador será alterado para moduladorQAM que baseado nos valores dos bits de entrada irá obter de uma tabela o valor da fase da respectiva quadratura de forma a poder realizar a operação:

$A*\cos(\phi)*\cos(2*\pi*f_o*t)-A*\sin(\phi)*\sin(2*\pi*f_o*t)$, cujo valor de Amplitude é a energia do tempo de bit respectivo.

A tabela é estipulada na seguinte:

Fase	Simbolo
0	000
45	001
90	011
135	111
180	101
225	100
270	110
315	010

Teremos que alterar igualmente o nosso receptor para interpretar os símbolos correctos do meio de transmissão.

Para o efeito teremos um correladorQAM que irá ter dois correladores (em seno e coseno) para descodificar o símbolo respectivo. Este modulo irá utilizar outro módulo modificado o decisorQAM que mediante dos valores obtidos no correlador irá retornar o respectivo símbolo. Este modulo irá utilizar a tabela de descodificação de símbolos indicada anteriormente.

Manteremos os módulos de detecção e correcção de erros desenvolvidos para responder ao solicitado.

É criado um módulo SCDQAM de forma a tornar simples a execução do sistema para a emissão/recepção dos ficheiros de texto e imagem.

Conclusão

Durante a elaboração tivemos oportunidade de colocar em prática os conhecimentos obtidos durante o semestre, contudo a implementação não foii pacífica. Muitos dos inconvenientes surgiram nos fracos conhecimentos do ambiente do MatLab que nos levantou muitas barreiras na implementação dos problemas propostos.

Apesar de termos consciência que, de forma geral, sabemos o conteúdo programático da cadeira, a sua implementação neste ambiente foi difícil no sentido que apesar do conteúdo que tínhamos interiorizado, muitas vezes, não conseguíamos escrever o código correspondente. Foram algumas as vezes que ficávamos 5 a 8 horas de volta do código e não nos era possível obter uma linha de código útil.

Agradecemos toda a ajuda disponibilizada pelo Eng. Artur Ferreira, que por diversas vezes nos “desbloqueava a passagem” indicando-nos alternativas à implementação.

No geral achamos que todo o processo de comunicação digital aliciante, que nos demonstrou o modo como o processo decorre, assim como as várias formas alternativas e as suas utilização.

Todos nós concordamos que os recursos disponibilizados foram úteis sendo complementares ao que foi dado nas aulas, assim como a forma como a matéria foi dada, de forma clara e quando havia dúvidas eram esclarecidas havendo um período de revisões da matéria da aula anterior.