

# 18-758 Project Report

Jean-Baptiste Cordonnier - Thomas Mullins

December 10, 2014

## A params.m

```

1 % signal timing
2 txSamplingFrequency = 100 * 10^6; % Hz
3 rxSamplingFrequency = 25 * 10^6; % Hz
4 symbolRate = 12.5 * 10^6; % Hz
5
6 % signal parameters
7 M = 4; % size of constellation
8 constellation = zeros(1,M);
9 for i = 0:M-1
10     constellation(i+1) = M.PSK_encode(de2bi(i, nextpow2(M)),M,1);
11 end
12 alpha = 0.25; % SRRC coefficient
13 txPad = 32; % extra 0 symbols to transmit on either side of ...
    message
14
15
16 pulseCenter = 500;
17
18 % receive parameters
19 rxUpsample = 4;
20
21 % transmit signal sync sequences
22 timingSync = [1 1 1 -0.4352 - 0.4398i -0.2497 + 0.5485i ...
    -0.0417 - 0.1385i -0.4626 + 0.6763i -0.1335 + 0.4968i ...
    0.1663 - 0.1779i 0.5440 + 0.4108i -0.0505 + 0.4528i ...
    0.5746 - 0.1020i -0.2389 + 0.1394i 0.5797 + 0.2914i ...
    -0.1767 + 0.3388i 0.6549 + 0.0617i 0.0578 - 0.2724i ...
    -0.6183 - 0.4586i -0.5870 - 0.0527i -0.7076 + 0.5985i ...
    0.2059 - 0.7190i -0.6773 - 0.4204i -0.0649 - 0.5375i ...
    -0.7086 + 0.3275i -0.2104 + 0.4044i -0.0914 - 0.0915i ...
    -0.6501 - 0.6495i -0.5897 + 0.1356i -0.3734 + 0.4923i ...
    0.5152 + 0.6686i -0.0160 - 0.4034i -0.3948 + 0.0531i ...
    0.3780 - 0.2198i -0.0559 + 0.2009i 0.6019 - 0.4881i ...
    0.3110 + 0.1121i -0.0962 + 0.5541i -0.1542 - 0.4630i ...
    0.1923 + 0.1788i -0.2481 + 0.4369i 0.7203 + 0.6936i ...
    -0.5379 - 0.3861i -0.6870 + 0.1549i -0.5613 - 0.1335i ...
    0.5539 + 0.0694i -0.1889 - 0.4206i -0.0852 + 0.6579i ...
    -0.5422 - 0.0422i 0.5147 - 0.6585i 0.2764 + 0.6908i ...
    -0.3126 - 0.5281i 0.2672 + 0.5905i 0.1599 + 0.5768i ...
    -0.4421 + 0.3669i -0.2217 - 0.1174i -0.4965 + 0.4600i ...
    0.1802 + 0.3440i 0.4400 - 0.6241i 0.6501 - 0.0035i ...
    0.3680 + 0.3496i 0.4775 - 0.4954i -0.0616 + 0.1703i ...
    0.6233 + 0.4832i 0.5703 + 0.1190i 0.1193 + 0.5119i ...
    -0.6708 + 0.5558i -0.1331 - 0.6686i 0.3550 - 0.4978i ...
    -0.5135 + 0.1528i -0.3541 - 0.2536i -0.1416 - 0.1350i ...
    -0.1641 + 0.1584i -0.4804 - 0.4498i -0.5846 - 0.2550i ...
    0.3888 - 0.3834i 0.3466 + 0.2781i 0.4674 + 0.4730i ...
    -0.2980 - 0.2749i 0.0332 - 0.2519i 0.4786 + 0.4475i ...
    0.0822 - 0.3418i 0.2604 - 0.3841i -0.0628 - 0.1665i ...
    0.0557 + 0.7091i 0.3681 + 0.6929i -0.3825 + 0.0412i ...
    -0.6469 + 0.3705i 0.1471 + 0.5151i 0.7042 + 0.6194i ...
    -0.1305 - 0.7206i 0.0590 - 0.4215i -0.4048 - 0.2512i ...
    -0.5827 + 0.3570i 0.3584 + 0.0624i -0.2334 + 0.4793i ...

```

```

    0.0758 + 0.6598i    0.5665 - 0.2069i    0.0669 - 0.2211i    ...
    0.1771 + 0.4278i    0.3546 - 0.5400i    0.4649 - 0.6848i    ...
    -0.1234 + 0.3337i    0.4058 - 0.1914i    ones(1,10) zeros(1,10)];
23 pilot = ones(1, 5);
24
25 %imageDimension = [66 46];
26 % imageDimension = [97 68];
27 imageDimension = [140 98];
28 imageSize = imageDimension(1) * imageDimension(2);
29 imageFile = strcat('images/shannon', int2str(imageSize), '.bmp');
30 txImage = imread(imageFile);
31 txMessageBits = reshape(txImage, [1 imageSize]);
32 txMessageBits = [ txMessageBits 0 0];
33
34 % channel coding parameters
35 % rate 1/3
36 %g = oct2dec([367 331 225]);
37 %nu = 7;
38 % rate 2/3
39 % g = oct2dec([34 31 17; 3 16 15]);
40 % nu = 7;
41 % rate 3/4
42 g = oct2dec([6,1,4,3;3,4,0,7;2,6,7,1]);
43 nu = 6;
44
45 txCodedBits = channelEncode(txMessageBits, g, nu);
46
47 messageSizeBits = length(txCodedBits);
48 messageSizeSymb = messageSizeBits / nextpow2(M);
49 packetSizeInfo = 120; % information symbols
50 packetSizeTot = length(pilot) + packetSizeInfo;

```

## B receiver.m

```

1 params;
2 Fs = rxSamplingFrequency * rxUpsample; % Hz
3 n = Fs / symbolRate; % samples per symbol
4
5 load('receivedsignal.mat');
6
7 receivedsignal = resample(receivedsignal, rxUpsample, 1);
8 plotSignal(receivedsignal, Fs);
9
10 % start = find(abs(receivedsignal) > 0.075, 1);
11 % receivedsignal = receivedsignal(start:length(receivedsignal));
12
13 % Time recovery to determine tau hat
14 tau.hat = doTimingSync(receivedsignal, timingSync, n, alpha);
15
16 % sampling
17 nSample = length(timingSync) + packetSizeTot * ...
    ceil(messageSizeSymb / packetSizeInfo);

```

```

18 samples = doSampling(receivedsignal, nSample, ...
    txSamplingFrequency / symbolRate, tau.hat);
19 cutSamples = samples((length(timingSync) + 2) : length(samples));
20
21 figure;
22 t = 1:length(cutSamples);
23 plot(t, real(cutSamples), 'b', t, imag(cutSamples), 'r')
24 title('cutSamples')
25
26 % Equalization and pilot removal
27 nSegments = floor(messageSizeSymb / packetSizeInfo);
28 messageSymbols = zeros(1, messageSizeSymb);
29 samp = ((1:nSegments+1)-1) * packetSizeTot + 1;
30 mess = ((1:nSegments+1)-1) * packetSizeInfo + 1;
31 for i = 1:nSegments
32     s = cutSamples(samp(i) : samp(i)+packetSizeTot-1);
33     eqSamples = equalize(pilot, s);
34     messageSymbols(mess(i):mess(i) + packetSizeInfo - 1) = ...
        eqSamples;
35 end
36
37 remainingSamples = mod(messageSizeSymb, packetSizeInfo);
38 if remainingSamples ≠ 0
39     samples = samples(samp(nSegments + 1) : samp(nSegments + 1) ...
        + length(pilot) + remainingSamples - 1);
40     eqSamples = equalize(pilot, samples);
41     messageSymbols(mess(nSegments + 1):mess(nSegments + 1) + ...
        remainingSamples - 1) = eqSamples;
42 end
43
44 % Constellation before and after equalization plots
45 figure;
46 subplot(1,2,1)
47 endMessage = find(abs(samples) > 0.075, 5, 'last');
48 plot(real(samples(1:endMessage)), imag(samples(1:endMessage)), ...
    'bo', real(constellation), imag(constellation), 'r*');
49 title('Samples before equalization')
50 subplot(1,2,2)
51 plot(real(messageSymbols), imag(messageSymbols), 'bo', ...
    real(constellation), imag(constellation), 'r*');
52 title('Samples after equalization')
53
54 % symbols to bit decoding
55 rxCodedBits = M.PSK_decode(messageSymbols, M);
56 rxMessageBits = channelDecode(rxCodedBits, g, nu);
57
58 codedBER = sum(rxCodedBits ≠ txCodedBits) / length(rxCodedBits);
59 BER = sum(rxMessageBits ≠ txMessageBits) / length(rxMessageBits);
60 fprintf('Coded BER = %f\n', codedBER);
61 fprintf('BER = %f\n', BER);
62
63 % show image
64 rxMessageBits = rxMessageBits(1:length(rxMessageBits)-2);
65 rxImage = reshape(rxMessageBits, imageDimension);
66 figure
67 subplot(1,3,1)
68 imshow(txImage)

```

```

69 title('Image transmitted')
70 subplot(1,3,2)
71 imshow(rxImage)
72 title('Image received')
73 subplot(1,3,3)
74 imshow(1-abs(rxImage-txImage))
75 title('Pixel errors')

```

## C transmitter.m

```

1  params;
2  n = txSamplingFrequency / symbolRate;
3
4  nSegments = ceil(messageSizeSymb / packetSizeInfo);
5  messageSymb = zeros(1, nSegments * length(pilot) + messageSizeSymb);
6  mb = ((1:nSegments) - 1) * packetSizeInfo * nextpow2(M) + 1;
7  ms = ((1:nSegments) - 1) * packetSizeTot + 1;
8  for i = 1:nSegments
9      symbols = ...
          M_PSK_encode(txCodedBits(mb(i):min(length(txCodedBits), ...
          mb(i) + packetSizeInfo * nextpow2(M) - 1)), M, 1);
10     messageSymb(ms(i) : min(length(messageSymb), ms(i) + ...
          packetSizeTot - 1)) = [ pilot symbols ];
11 end
12
13 symbols = [ones(1, 200) timingSync messageSymb];
14
15 fprintf('Transmitting message of %d bits in %d us\n', ...
          messageSizeBits, ceil(length(symbols) / symbolRate * 10^6));
16 if length(transmitsignal) > 800 * 10^-6 * txSamplingFrequency
17     error('Signal is too long (%d samples)\n', ...
          length(transmitsignal));
18 end
19
20
21 pulseTx = srrc(-pulseCenter:pulseCenter, alpha, n);
22 padding = zeros(1, txPad);
23 X = applyPulse([padding symbols padding], pulseTx, ...
          pulseCenterTx, n);
24 X = 0.9 * X / max(abs(X));
25
26 plotSignal(X, txSamplingFrequency);
27
28 transmitsignal = X;
29 save('transmitsignal.mat', 'transmitsignal');

```

## D doTimingSync.m

```

1  function tau.hat = doTimingSync(sign, timingSync, T, alpha)
2

```

```

3 % Create the centered pulse
4 pulseRx = srrc(-pulseCenter:pulseCenter, alpha, T);
5
6 % calculate expected timing frame
7 timingSync = applyPulse(timingSync, pulseRx, pulseCenterRx, T);
8
9 % correlate that with entire signal
10 [C, lag] = xcorr(sign, timingSync);
11
12 % plot it
13 figure;
14 plot(abs(C));
15 title('Convolution peaks in timing synchronisation');
16
17 % find the maximum correlation
18 [~, i_max] = max(C);
19 tau.hat = lag(i_max);
20
21 end

```

## E doSampling.m

```

1 function samples = doSampling(signal, nSamples, T.hat, tau.hat)
2
3     signal = signal(tau.hat:tau.hat+nSamples*T.hat-1);
4     preSamples = reshape(signal, T.hat, nSamples);
5     samples = preSamples(1,:);
6
7 end

```

## F applyPulse.m

```

1 function X = applyPulse(symbols, pulse, pulseCenter, n)
2
3     nSamples = n * (length(symbols) + 1);
4     X = zeros(1, nSamples);
5
6     paddedPulse = [ zeros(1, nSamples) pulse zeros(1, nSamples) ];
7     for i = 1:length(symbols)
8         t = n*i;
9         start = nSamples + pulseCenter - t;
10        pulseI = paddedPulse(start:start+nSamples-1);
11        X = X + symbols(i) * pulseI;
12    end
13
14 end

```

## G channelEncode.m

```
1 function Y = channelEncode(X, g, nu)
2 % X The information bits
3 % Y The coded bits
4 % g The table describing the trellis
5 % nu The number of bits of past state
6
7     k = size(g, 1);
8     n = size(g, 2);
9
10    nPastBits = ceil(nu/k);
11
12    % convert g to a 3-dimensional array:
13    % dim 1: k
14    % dim 2: nPastBits+1, one for each bit in octal number
15    % dim 3: n
16    G = [];
17    for i = 1:n
18        G = cat(3, G, de2bi(g(:, i), nPastBits+1));
19    end
20
21    % ensure we have k rows to group bits by stage
22    X = reshape(X, k, []);
23
24    % add some zero padding so we finish in the first state
25    X = [X zeros(k, nu + 1)];
26
27    Y = zeros(1, size(X, 2)*n);
28    Yi = 1;
29
30    % for each column (size k), emit n coded bits
31    state = zeros(k, nPastBits+1);
32    for info = X
33        % push info into front of state
34        state = [info state(:, 1:nPastBits)];
35        % for each output bit, do "xor" computation
36        for i = 1:n
37            andResult = G(:, :, i) .* state; % "and"
38            Y(Yi) = mod(sum(sum(andResult)), 2); % "xor"
39            Yi = Yi + 1;
40        end
41    end
42
43 end
```

## H channelDecode.m

```
1 function Y = channelDecode(X, g, nu)
2 % X The coded bits
3 % Y The information bits
```

```

4 % g The table describing the trellis
5 % nu The number of bits of past state
6
7 k = size(g, 1);
8 n = size(g, 2);
9
10 nStates = 2^nu;
11 nInputs = 2^k;
12 nPastBits = ceil(nu/k);
13
14 % convert g to a 3-dimensional array:
15 % dim 1: k
16 % dim 2: nPastBits+1, one for each bit in octal number
17 % dim 3: n
18 G = [];
19 for i = 1:n
20     G = cat(3, G, de2bi(g(:, i), nPastBits+1));
21 end
22
23 % ensure we have n rows to group bits by stage
24 X = reshape(X, n, []);
25 Xlen = size(X, 2);
26
27 % generate table of state transitions and codes ahead of time
28 nextStateTable = zeros(nStates, nInputs);
29 codedTable = zeros(n, nStates, nInputs);
30 for state = 0:nStates-1
31     % zero fill binState to ensure it can be reshaped into k ...
32     rows
33     binState = [de2bi(state, nu) zeros(1, mod(nu, k))];
34     binState = reshape(binState, k, []);
35     for input = 0:nInputs-1
36         binInput = de2bi(input, k);
37         for i = 1:n
38             andResult = G(:, :, i) .* [binInput' binState]; ...
39             % "and"
40             codedTable(i, state+1, input+1) = ...
41             mod(sum(sum(andResult)), 2); % "xor"
42         end
43         nextState = [binInput' binState(:, 1:nPastBits-1)];
44         nextState = reshape(nextState, 1, []);
45         nextStateTable(state+1, input+1) = ...
46         bi2de(nextState(1:nu));
47     end
48 end
49
50 dist = zeros(nStates, Xlen+1) + Inf;
51 dist(1, 1) = 0;
52 prevState = zeros(nStates, Xlen+1);
53 prevInput = zeros(nStates, Xlen+1);
54
55 % move forward through stages, filling dist, prevState, and ...
56 % prevInput
57 for Xi = 1:Xlen
58     % get a column vector of received bits, repeated for ...
59     % each state
60     coded = repmat(X(:, Xi), 1, nStates);

```



```

56     for input = 0:nInputs-1
57         % get table of (coded bits, states)
58         idealCoded = codedTable(:, :, input+1);
59         % get row vector of next states
60         nextStates = nextStateTable(:, input+1)';
61         % compute difference for each state
62         diff = sum(abs(idealCoded - coded))';
63         totalDists = dist(:, Xi) + diff;
64         for state = 1:nStates
65             nextState = nextStates(state)+1;
66             if totalDists(state) < dist(nextState, Xi+1)
67                 dist(nextState, Xi+1) = totalDists(state);
68                 prevState(nextState, Xi+1) = state-1;
69                 prevInput(nextState, Xi+1) = input;
70             end
71         end
72     end
73 end
74
75 % move backward through stages, tracking min distance
76 state = 0;
77 Y = zeros(1, Xlen*k);
78 for Xi = Xlen:-1:1
79     input = prevInput(state+1, Xi+1);
80     Y((Xi-1)*k + 1 : Xi*k) = de2bi(input, k);
81     state = prevState(state+1, Xi+1);
82 end
83
84 % remove 0 padding from encoding
85 Y = Y(1:length(Y)-(nu+1)*k);
86
87 end

```

## I plotSignal.m

```

1 function plotSignal(X, Fs)
2 %PLOTSIGNAL Plots a complex signal in time and frequency domains
3 % X is the signal. Fs is the sample frequency in Hz.
4
5 [T, F, Y] = DTFT(X, Fs / 10^6, 0);
6
7 figure();
8
9 subplot(2, 1, 1);
10 plot(T, real(X), T, imag(X));
11 xlabel('Time (us)');
12 pan on;
13 zoom on;
14
15 rgnA = abs(F) > 11.25;
16 rgnB = abs(F) > 12.5;
17 rgnC = abs(F) > 35;
18 valA = (abs(F) - 11.25) * -40 / 1.25;

```

```

19     valB = (abs(F) - 12.5) * -30 / 22.5 - 40;
20     envelope = zeros(length(F), 1);
21     envelope(rgnA) = valA(rgnA);
22     envelope(rgnB) = valB(rgnB);
23     envelope(rgnC) = -70;
24
25     subplot(2, 1, 2);
26     plot(F, 10*log10(abs(Y)), F, envelope);
27     xlabel('Frequency (MHz)');
28     pan xon;
29     zoom xon;
30 end

```

## J DTFT.m

```

1 function [t, f, Z] = DTFT(z, Fs, n)
2 %DTFT Computes the DTFT of a signal with proper units
3
4     L = length(z);
5     L2 = pow2(n + nextpow2(L));
6     t = (0:L-1) / Fs;
7     f = (-L2/2:L2/2-1) * (Fs / L2);
8     Z = fftshift(fft(z, L2)) * 2 / L;
9
10 end

```