18-758 Project Report

Jean-Baptiste Cordonnier - Thomas Mullins

December 10, 2014

1 Pulse

2 Timing sync

We use a simplified version of the timing synchronization we have seen in class. We use a random sequence of symbols that is highly uncorrelated, such that we see a high peak during the correlation of the message and the timing synchronization (see plot). Then we get $\hat{\tau}$ and know where the signal starts exactly.

3 One-tap equalizer

We are using a one-tap equalizer for each segment of the message. We send 5-length symbols pilot every 120 symbols of information. The one tap equalizer uses the following formula :

$$h_0 = \frac{\text{txPilot} \cdot \text{rxPilot}}{\text{txPilot}^2} \qquad\qquad \text{eqMessage} = \frac{\text{message}}{h_0}$$

The equalization is very efficient as we can see on the following plot. Even if the pilot sequence is very short, the h_0 is still precise and equalize well the received segment of the message. It corrects the phase drift and the modulus of the signal.

4 Constellation

We are using a 4-PSK constelation.

5 Channel Coding

6 Conclusion

A params.m

```
% signal timing
txSamplingFrequency =  100 * 10^6; % Hz
rxSamplingFrequency =   25 * 10^6; % Hz
symbolRate          = 12.5 * 10^6; % Hz

% signal parameters
M = 4;        % size of constelation
constelation = zeros(1,M);
for i = 0:M-1
    constelation(i+1) = M_PSK_encode(de2bi(i, nextpow2(M)),M,1);
end
alpha = 0.25; % SRRC coefficient
txPad = 32;   % extra 0 symbols to transmit on either side of ...
    message


pulseCenter = 500;

% receive paramters
rxUpsample = 4;

% transmit signal sync sequences
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)];
pilot = ones(1, 5);

imageDimension = [140 98];
imageSize = imageDimension(1) * imageDimension(2);
imageFile = strcat('images/shannon', int2str(imageSize), '.bmp');
txImage = imread(imageFile);
txMessageBits = reshape(txImage, [1 imageSize]);
txMessageBits = [ txMessageBits 0 0];

% rate 3/4
g = oct2dec([6,1,4,3;3,4,0,7;2,6,7,1]);
nu = 6;

txCodedBits = channelEncode(txMessageBits, g, nu);

messageSizeBits = length(txCodedBits);
messageSizeSymb = messageSizeBits / nextpow2(M);
packetSizeInfo = 120; % information symbols
packetSizeTot = length(pilot) + packetSizeInfo;
```

B receiver.m

```
params;
Fs = rxSamplingFrequency * rxUpsample; % Hz
n = Fs / symbolRate; % samples per symbol

load('receivedsignal.mat');

receivedsignal = resample(receivedsignal, rxUpsample, 1);
plotSignal(receivedsignal, Fs);

% start = find(abs(receivedsignal) > 0.075, 1);
% receivedsignal = receivedsignal(start:length(receivedsignal));

% Time recovery to determine tau hat
tau_hat = doTimingSync(receivedsignal, timingSync, n, alpha);

% sampling
nSample = length(timingSync) + packetSizeTot * ...
    ceil(messageSizeSymb / packetSizeInfo);
samples = doSampling(receivedsignal, nSample, ...
    txSamplingFrequency / symbolRate, tau_hat);
cutSamples = samples((length(timingSync) + 2) : length(samples));

figure;
t = 1:length(cutSamples);
plot(t, real(cutSamples), 'b', t, imag(cutSamples), 'r')
title('cutSamples')

% Equalization and pilot removal
```

```
nSegments = floor(messageSizeSymb / packetSizeInfo);
messageSymbols = zeros(1, messageSizeSymb);
samp = ((1:nSegments+1)-1) * packetSizeTot + 1;
mess = ((1:nSegments+1)-1) * packetSizeInfo + 1;
for i = 1:nSegments
    s = cutSamples(samp(i) : samp(i)+packetSizeTot-1);
    eqSamples = equalize(pilot, s);
    messageSymbols(mess(i):mess(i) + packetSizeInfo - 1) = ...
        eqSamples;
end

remainingSamples = mod(messageSizeSymb, packetSizeInfo);
if remainingSamples ≠ 0
    samples = samples(samp(nSegments + 1) : samp(nSegments + 1) ...
        + length(pilot) + remainingSamples - 1);
    eqSamples = equalize(pilot, samples);
    messageSymbols(mess(nSegments + 1):mess(nSegments + 1) + ...
        remainingSamples - 1) = eqSamples;
end

% Constelation before and after equalization plots
figure;
subplot(1,2,1)
endMessage = find(abs(samples) > 0.075, 5, 'last');
plot(real(samples(1:endMessage)), imag(samples(1:endMessage)), ...
    'bo', real(constelation), imag(constelation), 'r*');
title('Samples before equalization')
subplot(1,2,2)
plot(real(messageSymbols), imag(messageSymbols), 'bo', ...
    real(constelation), imag(constelation), 'r*');
title('Samples after equalization')

% symobols to bit decoding
rxCodedBits = M_PSK_decode(messageSymbols, M);
rxMessageBits = channelDecode(rxCodedBits, g, nu);

codedBER = sum(rxCodedBits ≠ txCodedBits) / length(rxCodedBits);
BER = sum(rxMessageBits ≠ txMessageBits) / length(rxMessageBits);
fprintf('Coded BER = %f\n', codedBER);
fprintf('BER = %f\n', BER);

% show image
rxMessageBits = rxMessageBits(1:length(rxMessageBits)-2);
rxImage = reshape(rxMessageBits, imageDimension);
figure
subplot(1,3,1)
imshow(txImage)
title('Image transmitted')
subplot(1,3,2)
imshow(rxImage)
title('Image received')
subplot(1,3,3)
imshow(1-abs(rxImage-txImage))
title('Pixel errors')
```

C transmitter.m

```matlab
params;
n = txSamplingFrequency / symbolRate;

nSegments = ceil(messageSizeSymb / packetSizeInfo);
messageSymb = zeros(1, nSegments * length(pilot) + messageSizeSymb);
mb = ((1:nSegments) - 1) * packetSizeInfo * nextpow2(M) + 1;
ms = ((1:nSegments) - 1) * packetSizeTot + 1;
for i = 1:nSegments
   symbols = ...
        M_PSK_encode(txCodedBits(mb(i):min(length(txCodedBits), ...
        mb(i) + packetSizeInfo * nextpow2(M) - 1)), M, 1);
   messageSymb(ms(i) : min(length(messageSymb), ms(i) + ...
        packetSizeTot - 1))  = [ pilot symbols ];
end

symbols = [ones(1, 200) timingSync messageSymb];

fprintf('Transmitting message of %d bits in %d us\n', ...
    messageSizeBits, ceil(length(symbols) / symbolRate * 10^6));
if length(transmitsignal) > 800 * 10^-6 * txSamplingFrequency
    error('Signal is too long (%d samples)\n', ...
        length(transmitsignal));
end

pulseTx = srrc(-pulseCenter:pulseCenter, alpha, n);
padding = zeros(1, txPad);
X = applyPulse([padding symbols padding], pulseTx, ...
    pulseCenterTx, n);
X = 0.9 * X / max(abs(X));

plotSignal(X, txSamplingFrequency);

transmitsignal = X;
save('transmitsignal.mat', 'transmitsignal');
```

D doTimingSync.m

```matlab
function tau_hat = doTimingSync(sign, timingSync, T, alpha)

% Create the centered pulse
pulseRx = srrc(-pulseCenter:pulseCenter, alpha, T);

% calculate expected timing frame
timingSync = applyPulse(timingSync, pulseRx, pulseCenterRx, T);

% correlate that with entire signal
[C, lag] = xcorr(sign, timingSync);

% plot it
```

```
figure;
plot(abs(C));
title('Convolution peaks in timing synchronisation');

% find the maximum correlation
[¬, i_max] = max(C);
tau_hat = lag(i_max);

end
```

E doSampling.m

```
function samples = doSampling(signal, nSamples, T_hat, tau_hat)

    signal = signal(tau_hat:tau_hat+nSamples*T_hat-1);
    preSamples = reshape(signal, T_hat, nSamples);
    samples = preSamples(1,:);

end
```

F applyPulse.m

```
function X = applyPulse(symbols, pulse, pulseCenter, n)

    nSamples = n * (length(symbols) + 1);
    X = zeros(1, nSamples);

    paddedPulse = [ zeros(1, nSamples) pulse zeros(1, nSamples) ];
    for i = 1:length(symbols)
        t = n*i;
        start = nSamples + pulseCenter - t;
        pulseI = paddedPulse(start:start+nSamples-1);
        X = X + symbols(i) * pulseI;
    end

end
```

G channelEncode.m

```
function Y = channelEncode(X, g, nu)
% X The information bits
% Y The coded bits
% g  The table describing the trellis
% nu The number of bits of past state
```

```

    k = size(g, 1);
    n = size(g, 2);

    nPastBits = ceil(nu/k);

    % convert g to a 3-dimensional array:
    %  dim 1: k
    %  dim 2: nPastBits+1, one for each bit in octal number
    %  dim 3: n
    G = [];
    for i = 1:n
        G = cat(3, G, de2bi(g(:, i), nPastBits+1));
    end

    % ensure we have k rows to group bits by stage
    X = reshape(X, k, []);

    % add some zero padding so we finish in the first state
    X = [X zeros(k, nu + 1)];

    Y = zeros(1, size(X, 2)*n);
    Yi = 1;

    % for each column (size k), emit n coded bits
    state = zeros(k, nPastBits+1);
    for info = X
        % push info into front of state
        state = [info state(:,1:nPastBits)];
        % for each output bit, do "xor" computation
        for i = 1:n
            andResult = G(:, :, i) .* state; % "and"
            Y(Yi) = mod(sum(sum(andResult)), 2); % "xor"
            Yi = Yi + 1;
        end
    end

end
```

H channelDecode.m

```
function Y = channelDecode(X, g, nu)
% X  The coded bits
% Y  The information bits
% g  The table describing the trellis
% nu The number of bits of past state

    k = size(g, 1);
    n = size(g, 2);

    nStates = 2^nu;
    nInputs = 2^k;
    nPastBits = ceil(nu/k);
```

```

    % convert g to a 3-dimensional array:
    %  dim 1: k
    %  dim 2: nPastBits+1, one for each bit in octal number
    %  dim 3: n
    G = [];
    for i = 1:n
        G = cat(3, G, de2bi(g(:, i), nPastBits+1));
    end

    % ensure we have n rows to group bits by stage
    X = reshape(X, n, []);
    Xlen = size(X, 2);

    % generate table of state transitions and codes ahead of time
    nextStateTable = zeros(nStates, nInputs);
    codedTable = zeros(n, nStates, nInputs);
    for state = 0:nStates-1
        % zero fill binState to ensure it can be reshaped into k ...
            rows
        binState = [de2bi(state, nu) zeros(1, mod(nu, k))];
        binState = reshape(binState, k, []);
        for input = 0:nInputs-1
            binInput = de2bi(input, k);
            for i = 1:n
                andResult = G(:, :, i) .* [binInput' binState]; ...
                    % "and"
                codedTable(i, state+1, input+1) = ...
                    mod(sum(sum(andResult)), 2); % "xor"
            end
            nextState = [binInput' binState(:,1:nPastBits-1)];
            nextState = reshape(nextState, 1, []);
            nextStateTable(state+1, input+1) = ...
                bi2de(nextState(1:nu));
        end
    end

    dist = zeros(nStates, Xlen+1) + Inf;
    dist(1, 1) = 0;
    prevState = zeros(nStates, Xlen+1);
    prevInput = zeros(nStates, Xlen+1);

    % move forward through stages, filling dist, prevState, and ...
        prevInput
    for Xi = 1:Xlen
        % get a column vector of received bits, repeated for ...
            each state
        coded = repmat(X(:, Xi), 1, nStates);
        for input = 0:nInputs-1
            % get table of (coded bits, states)
            idealCoded = codedTable(:, :, input+1);
            % get row vector of next states
            nextStates = nextStateTable(:, input+1)';
            % compute difference for each state
            diff = sum(abs(idealCoded - coded))';
            totalDists = dist(:, Xi) + diff;
            for state = 1:nStates
```

```
                nextState = nextStates(state)+1;
                if totalDists(state) < dist(nextState, Xi+1)
                    dist(nextState, Xi+1) = totalDists(state);
                    prevState(nextState, Xi+1) = state-1;
                    prevInput(nextState, Xi+1) = input;
                end
            end
        end
    end

    % move backward through stages, tracking min distance
    state = 0;
    Y = zeros(1, Xlen*k);
    for Xi = Xlen:-1:1
        input = prevInput(state+1, Xi+1);
        Y((Xi-1)*k + 1 : Xi*k) = de2bi(input, k);
        state = prevState(state+1, Xi+1);
    end

    % remove 0 padding from encoding
    Y = Y(1:length(Y)-(nu+1)*k);

end
```

I plotSignal.m

```
function plotSignal(X, Fs)
%PLOTSIGNAL Plots a complex signal in time and frequency domains
%   X is the signal. Fs is the sample frequency in Hz.

    [T, F, Y] = DTFT(X, Fs / 10^6, 0);

    figure();

    subplot(2, 1, 1);
    plot(T, real(X), T, imag(X));
    xlabel('Time (us)');
    pan xon;
    zoom xon;

    rgnA = abs(F) > 11.25;
    rgnB = abs(F) > 12.5;
    rgnC = abs(F) > 35;
    valA = (abs(F) - 11.25) * -40 / 1.25;
    valB = (abs(F) - 12.5) * -30 / 22.5 - 40;
    envelope = zeros(length(F), 1);
    envelope(rgnA) = valA(rgnA);
    envelope(rgnB) = valB(rgnB);
    envelope(rgnC) = -70;

    subplot(2, 1, 2);
    plot(F, 10*log10(abs(Y)), F, envelope);
    xlabel('Frequency (MHz)');
```

```
    pan xon;
    zoom xon;
end
```

J DTFT.m

```
function [t, f, Z] = DTFT(z, Fs, n)
%DTFT Computes the DTFT of a signal with proper units

    L = length(z);
    L2 = pow2(n + nextpow2(L));
    t = (0:L-1) / Fs;
    f = (-L2/2:L2/2-1) * (Fs / L2);
    Z = fftshift(fft(z, L2)) * 2 / L;

end
```