EE451 PROJECT PROGRESS REPORT 3

Students: Harun Durmuş **Student ID's:** 270206025 **Date:** 21.12.2023

Mustafa Eren Çelebi 270206053

Necmettin Yiğit Dübek 260206049

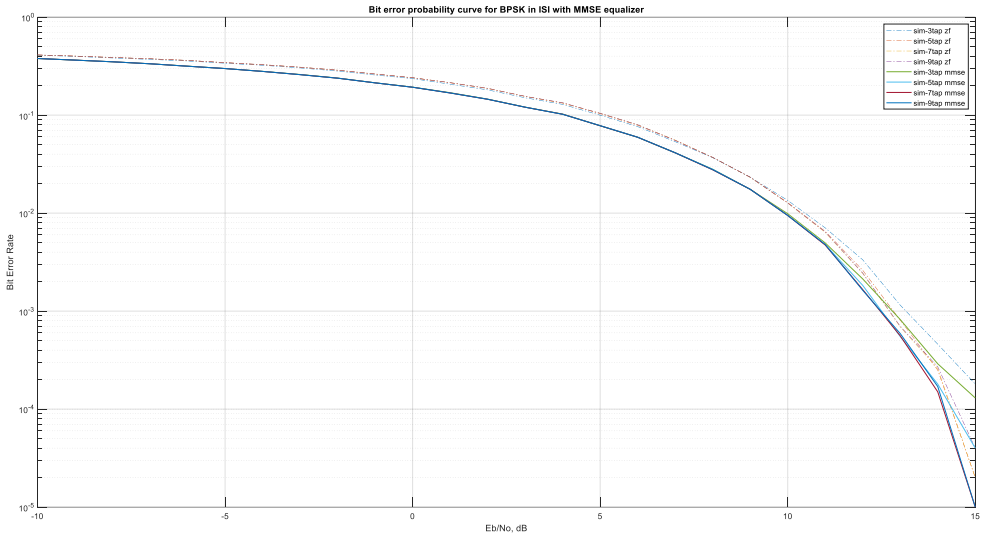
Week 12

This week, we perform the Huffman source coding[1] into the audio file that we want to use. However, we have encountered some problems about the very large matrix sizes. We tried to apply 8-PSK modulation into the encoded data. Our mentor Aslı Taşçı has analyzed the code and suggested us about not to use 8-PSK because of the too much computation we would deal with.

Then, we created an alternative method[2] to succeed in the project by applying B-PSK modulation into the random bit data we created ourself. Zero-Forcing and MMSE Equalization methods are applied to the signal. In a real-world BPSK system, the binary data (after being mapped to +1 and -1) is indeed multiplied by a carrier signal. This carrier is usually a high-frequency sinusoidal wave. Because of the simplicity of the data to be processed, carrier multiplication is not applied. For the sake of analysis, especially when focusing on topics like BER performance in the presence of noise or ISI, the carrier component may be omitted to simplify the mathematics. The focus in such cases is often on the baseband signal processing aspects.

BER Analysis: Bit Error Rate (BER) analysis is a critical metric in digital communication systems that counts the fraction of sent bits that are incorrectly received. It is calculated as the ratio of wrongly received bits to total broadcast bits during a certain period. BER is a crucial measure of the quality and dependability of a communication link that is directly influenced by elements such as signal-to-noise ratio (SNR), interference, modulation schemes, and the availability of error correction mechanisms. Essentially, it provides a numerical assessment of a system's performance, assisting engineers and designers in understanding the effectiveness of their communication protocols and hardware under varied operational scenarios. Lower BER values indicate greater system performance, reflecting improved data transmission accuracy and integrity.

In summary, we applied the different equalization with different tap values on the AWGN channels and applied BER analysis for different SNR values.



Note: The analysis of the results and literature search is going to be made in the lab session. Results are going to be compared with the researchments that made in the beginning of the project.

Appendices

[1] MATLAB Code "p3_week11.m"

```
clc;
clear all;
close all;
%% RUN ETMESİ UZUN SÜRÜYOR. mod_psk 1x1843200 uzunluğunda olduğu için (line 142)
%% reading audio file
[x,fs] = audioread('handel.wav');
N = length(x);
%% quantization
b = max(x);
a = min(x);
Nq = 3; % quantization number
quantized = floor(((x-a)/(b-a))*(2^Nq-1))*((b-a)/(2^Nq-1)) + a;
mx = max(quantized)/2;

figure (1)
subplot(211)
plot(x);
subplot(212)
plot(quantized);
% sound(quantized,fs);
% sound(x,fs);
%% huffman source-coding theorem
inputs = unique(quantized);
occurances = zeros(1,8);
for i=1:length(inputs)
    for j=1:length(quantized)
        if(inputs(i)==quantized(j))
            occurances(i) = occurances(i) +1;
```

```
        end
    end
end
probabilities = transpose(occurances./N);
% Input -0.800018310546875: Huffman Code - 0101001
% Input -0.571446010044643: Huffman Code - 01011
% Input -0.342873709542411: Huffman Code - 00
% Input -0.114301409040179: Huffman Code - 1
% Input 0.114270891462054:  Huffman Code - 011
% Input 0.342843191964286:  Huffman Code - 0100
% Input 0.571415492466518:  Huffman Code - 010101
% Input 0.799987792968750:  Huffman Code - 0101000
%% açıklama
% Huffman codelarını chat gpt ile hesaplattık. probabilityleri bulduk
% leveller belli. bunları gptye verdik hesapla dedik.
%% encoding audio signal
encoded = [];
for j=1:length(quantized)
    if(quantized(j)==inputs(1))
        encoded = [encoded 0 1 0 1 0 0 1];
    end
    if(quantized(j)==inputs(2))
        encoded = [encoded 0 1 0 1 1];
    end
    if(quantized(j)==inputs(3))
        encoded = [encoded 0 0];
    end
    if(quantized(j)==inputs(4))
        encoded = [encoded 1];
    end
    if(quantized(j)==inputs(5))
        encoded = [encoded 0 1 1];
    end
    if(quantized(j)==inputs(6))
        encoded = [encoded 0 1 0 0];
    end
    if(quantized(j)==inputs(7))
        encoded = [encoded 0 1 0 1 0 1];
    end
    if(quantized(j)==inputs(8))
        encoded = [encoded 0 1 0 1 0 0 0];
    end
end
figure(2)
plot(encoded);
%% defining carrier
fc = fs*2; % fc fs'ten büyük olması lazım ama mod_psk çok büyük olmadığı için x2
yaptık sadece
t = linspace(0,2*pi);
c1 = cos(2*pi*fc*t);
c2 = cos(2*pi*fc*t + pi/4);
c3 = cos(2*pi*fc*t + 2*pi/4);
c4 = cos(2*pi*fc*t + 3*pi/4);
c5 = cos(2*pi*fc*t + 4*pi/4);
c6 = cos(2*pi*fc*t + 5*pi/4);
c7 = cos(2*pi*fc*t + 6*pi/4);
c8 = cos(2*pi*fc*t + 7*pi/4);

% carrier = [c1, c2, c3, c4, c5, c6, c7, c8];
```

```
figure (3)
subplot(421)
plot(c1);
grid on;
subtitle("carrier c1(t)");

subplot(422)
plot(c2);
grid on;
subtitle("carrier c2(t)");

subplot(423)
plot(c3);
grid on;
subtitle("carrier c3(t)");

subplot(424)
plot(c4);
grid on;
subtitle("carrier c4(t)");

subplot(425)
plot(c5);
grid on;
subtitle("carrier c5(t)");

subplot(426)
grid on;
plot(c6);
subtitle("carrier c6(t)");

subplot(427)
plot(c7);
grid on;
subtitle("carrier c7(t)");

subplot(428)
plot(c8);
grid on;
subtitle("carrier c8(t)");
%% modulation
% Input -0.800018310546875: Huffman Code - 0101001
% Input -0.571446010044643: Huffman Code - 01011
% Input -0.342873709542411: Huffman Code - 00
% Input -0.114301409040179: Huffman Code - 1
% Input 0.114270891462054:  Huffman Code - 011
% Input 0.342843191964286:  Huffman Code - 0100
% Input 0.571415492466518:  Huffman Code - 010101
% Input 0.799987792968750:  Huffman Code - 0101000
%% açıklama
% aşağıda yaptığımız şey, her bir encoded sequence (huffman codeları) aynı
uzunlukta olmadığı için quantized sinyalde ilgili
% leveli gördüğü zaman ilgili sequence ile ilgili carrierı çarparak modüle
% etmek ve uç uca eklemek, aslında serial to parallel conversion yapıp
% sonra carrierlarla çarpıp tekrar birleştirmiş olduk
%%
i=1;
mod_psk = [];
```

```
for k=1:N
    if(quantized(k)==inputs(1))
        temp_modulation = transpose(encoded(i:i+6)).*c1;
        mod_psk = [mod_psk temp_modulation(1,:)];
        i = i + 7;
        temp_modulation = 0;
    end
    if(quantized(k)==inputs(2))
        temp_modulation = transpose(encoded(i:i+4)).*c2;
        mod_psk = [mod_psk temp_modulation(1,:)];
        i = i + 5;
        temp_modulation = 0;
    end
    if(quantized(k)==inputs(3))
        temp_modulation = transpose(encoded(i:i+1)).*c3;
        mod_psk = [mod_psk temp_modulation(1,:)];
        i = i + 2;
        temp_modulation = 0;
    end
    if(quantized(k)==inputs(4))
        temp_modulation = transpose(encoded(i)).*c4;
        mod_psk = [mod_psk temp_modulation(1,:)];
        i = i + 1;
        temp_modulation = 0;
    end
    if(quantized(k)==inputs(5))
        temp_modulation = transpose(encoded(i:i+2)).*c5;
        mod_psk = [mod_psk temp_modulation(1,:)];
        i = i + 3;
        temp_modulation = 0;
    end
    if(quantized(k)==inputs(6))
        temp_modulation = transpose(encoded(i:i+3)).*c6;
        mod_psk = [mod_psk temp_modulation(1,:)];
        i = i + 4;
        temp_modulation = 0;
    end
    if(quantized(k)==inputs(7))
        temp_modulation = transpose(encoded(i:i+5)).*c7;
        mod_psk = [mod_psk temp_modulation(1,:)];
        i = i + 6;
        temp_modulation = 0;
    end
    if(quantized(k)==inputs(8))
        temp_modulation = transpose(encoded(i:i+6)).*c8;
        mod_psk = [mod_psk temp_modulation(1,:)];
        i = i + 6;
        temp_modulation = 0;
    end
end
figure (4)
plot(mod_psk);
```

[2] MATLAB Code “w12_1.m”

```
clc;
clear all;
close all;
%%
Eb_N0_dB = -10:15; % multiple Eb/N0 values
nTAP = 4;
for i = 1:length(Eb_N0_dB)
    bits = randi([0 1],1,100000);
    N_bits = length(bits);
    % R = 1000; % Bit rate in bits per second
    % tb = 1/R; % Bit time
    % fc = 10 * R; % Carrier frequency, for example, 10 times the bit rate
    % fs = 4 * fc; % Sampling frequency, 4 times the carrier frequency
    % ts = 1/fs;
    % time = 0:ts:(N_bits*tb)-ts;
    %% modulation
    %%
    mxsig = 2*bits-1;
    % %% pulse shaping for carrier multiplication
    % %%
    % bits_reshaped = reshape(bits, N_bits, 1);
    % spb = tb*fs; % sample per bit
    % message = repmat(bits_reshaped, 1, spb);
    % message = reshape(message', 1, []);
    % %% defining carrier
    % %%
    % carrier = sin(2*pi*fc*time);
    % %% multiplication
    % %%
    % mxsig = message.*carrier;
    % %% figures
    % %%
    % figure(1)
    % subplot(311)
    % plot(time,message);
    % subplot(312)
    % plot(time,carrier);
    % subplot(313)
    % plot(time,mxsig);
    %% channel model (multipath channel)
    %%
    channel_response = [0.2 0.9 0.3];
    chan_out = conv(mxsig,channel_response);
    %% noise addition
    %%
    pavg_channel = sum(abs(chan_out).^2)/length(chan_out);
    snr_lin = 10^(0.1*Eb_N0_dB(i));
    var_noise = pavg_channel/snr_lin;
    noise = sqrt(var_noise)*randn(1,length(chan_out));
    noisy_out = chan_out + noise; % additive white gaussian noise
    %% equalization
    %%
    for k = 1:nTAP
        L  = length(channel_response);
        %% zero forcing equalizer
        %%
```

```
         channel_matrix = toeplitz([channel_response(2:end) zeros(1,2*k+1-L+1)], [
channel_response(2:-1:1) zeros(1,2*k+1-L+1) ]);
         d  = zeros(1,2*k+1);
         d(k+1) = 1;
         channel_zf  = (inv(channel_matrix)*d.').';
         %%  matched filter
         yFilt_zf = conv(noisy_out,channel_zf);
         yFilt_zf = yFilt_zf(k+2:end);
         yFilt_zf = conv(yFilt_zf,ones(1,1)); % convolution
         ySamp_zf = yFilt_zf(1:1:N_bits);  % sampling at time T
         %% receiver - hard decision decoding
         ipHat_zf = real(ySamp_zf)>0;
         %% counting the errors
         nErr_zf(k,i) = size(find(bits- ipHat_zf),2);
        %% mmse equalizer
        %%
         hAutoCorr = conv(channel_response,fliplr(channel_response));
         channel_matrix = toeplitz([hAutoCorr([3:end]) zeros(1,2*k+1-L)], [
hAutoCorr([3:end]) zeros(1,2*k+1-L) ]);
         channel_matrix = channel_matrix + 1/2*10^(-Eb_N0_dB(i)/10)*eye(2*k+1);
         d  = zeros(1,2*k+1);
         d([-1:1]+k+1) = fliplr(channel_response);
         channel_mmse  = [inv(channel_matrix)*d.'].';
         %%  matched filter
         yFilt_mmse = conv(noisy_out,channel_mmse);
         yFilt_mmse = yFilt_mmse(k+2:end);
         yFilt_mmse = conv(yFilt_mmse,ones(1,1)); % convolution
         ySamp_mmse = yFilt_mmse(1:1:N_bits);  % sampling at time T
         %% receiver - hard decision decoding
         ipHat_mmse = real(ySamp_mmse)>0;
         %% counting the errors
         nErr_mmse(k,i) = size(find(bits- ipHat_mmse),2);
     end
end
simBer_zf = nErr_zf/N_bits; % simulated ber
simBer_mmse = nErr_mmse/N_bits; % simulated ber
theoryBer = 0.5*erfc(sqrt(10.^(Eb_N0_dB/10))); % theoretical ber
figure
semilogy(Eb_N0_dB,simBer_zf(1,:),'-.');
hold on
semilogy(Eb_N0_dB,simBer_zf(2,:),'-.');
semilogy(Eb_N0_dB,simBer_zf(3,:),'-.');
semilogy(Eb_N0_dB,simBer_zf(4,:),'-.');
semilogy(Eb_N0_dB,simBer_mmse(1,:));
semilogy(Eb_N0_dB,simBer_mmse(2,:));
semilogy(Eb_N0_dB,simBer_mmse(3,:));
semilogy(Eb_N0_dB,simBer_mmse(4,:));
% axis([0 10 10^-3 0.5])
grid on
legend('sim-3tap zf', 'sim-5tap zf','sim-7tap zf','sim-9tap zf','sim-3tap mmse',
'sim-5tap mmse','sim-7tap mmse','sim-9tap mmse');
xlabel('Eb/No, dB');
ylabel('Bit Error Rate');
title('Bit error probability curve for BPSK in ISI with MMSE equalizer');
```