EQ2310 Digital Communications - Project Assignment

Milosh Jankovikj, Hareekeshav Sethumadhavan Srinivasan

I. Abstract

This report discusses the simulation outcomes and analysis of a QPSK system. The study involved creating Matlab functions and running multiple simulation tests to obtain the results. The tests examined various aspects of the system, including BER estimations, signal constellations given different E_b/N_o values, power spectra of different pulse shapes, eye diagrams, and the performance of phase estimation and synchronization for various lengths of the training sequence.

II. Background and Problem Formulation

The QPSK system comprises a transmitter, a channel, and a receiver. In this simulation, we create a set of training bits that undergo modulation with QPSK and pulse shaping. The transmitter's output is then transmitted to the channel, where white Gaussian noise is incorporated into the signal. After receiving the signal, it undergoes bandpass filtering, down-conversion, and matched filtering. The signal is then synchronized, and its phase is estimated. Ultimately, the detector functions as a threshold device, and a decision variable is determined at the output.

We have solved the missing functions and addressed the following problem areas:

- Investigating the BER performance as a function of E_b/N_o with perfect phase estimation and synchronization.
- Study about the signal constellation in the receiver for various values of E_b/N_o.
- Plotting the power spectral density for different pulse shapes.
- Plotting the eye diagram for the AWGN channel and the simple two-path ISI channel.
- Investigating the performance of phase estimation and synchronization for different lengths of the training sequence.

III. Methodology

1) *First problem area*
   After completing the four missing Matlab functions, the simulation code was enabled to test the simulated BER. The results were compared with the derived expression for the theoretical BER.
2) *Second problem area*
   Following the completed code, we focused on examining the signal constellation in the receiver for various SNR values. The dependence of the noise level with the constellation position and the phase estimate are looked at in the Results part.
3) *Third problem area*
   The rectangular pulse and the Root raised cosine pulse are plotted using the periodogram function to analyze the effect of different pulse shapes during the transmission phase.
4) *Fourth problem area*
   To compare the eye diagrams for the AWGN channel and Two-path ISI channel and their implications on the synchronization error and phase estimation accuracy, a simple multi-path channel along with the given AWGN channel is simulated, and the variance for each is calculated.

5) *Fifth problem area*

The last task of this report simulates the phase and synchronization error for various training sequences given a range of E_b/N_0 values. One can conclude an acceptable value given the performance of the phase and synchronization. Consequently, the overhead of using such training sequences is computed.

IV. Results

1) *BER*

This calculation determines the estimated error of the signal transmitted through the AWGN channel. The simulation's BER value corresponds closely with the theoretical BER, as determined by the following formula:

$$\text{BER} = Q\left(\sqrt{\frac{2E_b}{N_o}}\right) \tag{1}$$

During the simulation, the SNR was varied between 0 dB and 10 dB. As the SNR ratio increases, the BER decreases accordingly. Figure 1 displays the results of this correlation. The figure also displays the close agreement between the theoretical and estimated BER.

2) *Signal Constellation*

In this context, we analyze various SNRs mapped onto the QPSK constellation. Figure 2 showcases the outcomes of this mapping. Understanding that the noise level and SNR are directly proportional is crucial. As the noise level increases, the constellation will shift away from its center, resulting in more dispersion. The phase estimation error is caused by the AWGN channel and receiver filters, which introduce phase offset. As a result, offset estimation is critical for accurate demodulation at the receiver.

3) *Power Spectral Densities*

We utilized the MATLAB "periodogram()" function to generate spectra for Periodograms of both the rectangular pulse and the root-raised cosine. Theoretical analysis has shown that while the rectangular pulse does not efficiently eliminate ISI, the root-raised cosine does. By restricting occupied bandwidth and utilizing pulse shapes that meet the Nyquist ISI criterion, the root-raised cosine can successfully avoid ISI, as observed in the impulse response's time domain view. Figure 3 displays the power spectral densities for both the rectangular pulse and the root-raised cosine, with the latter displaying a narrower bandwidth due to eliminating high-frequency components and thus reducing the likelihood of ISI.

4) *AWGN and two-path ISI channel eye diagram*

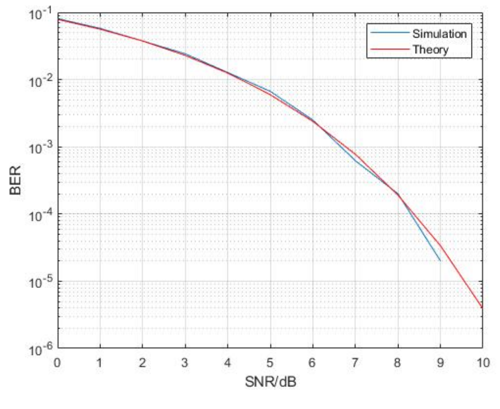
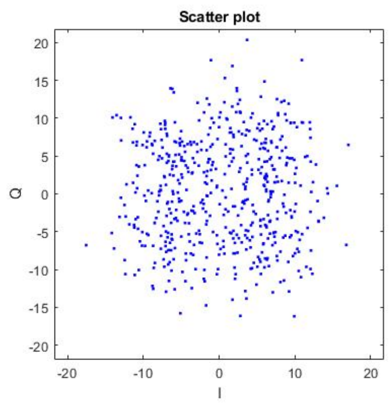
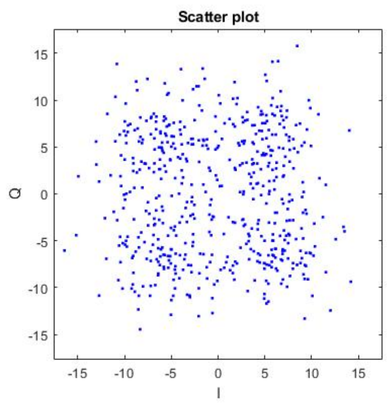


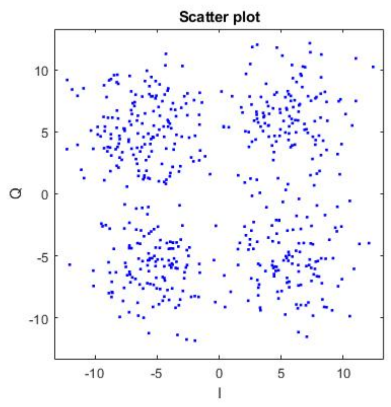
Fig. 1: Theoretical vs. Simulated BER.



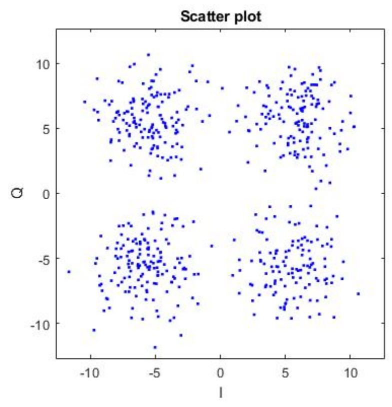
(a) $E_b/N_0 = 0$ dB



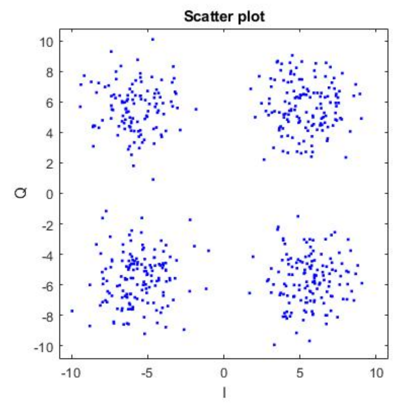
(b) $E_b/N_0 = 2$ dB



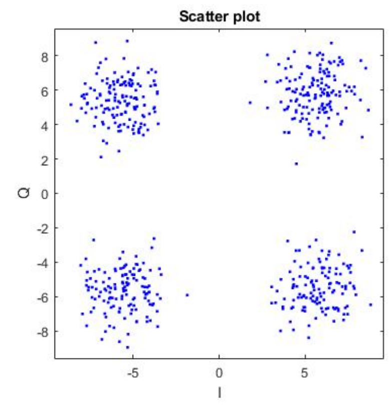
(c) $E_b/N_0 = 4$ dB



(d) $E_b/N_0 = 6$ dB

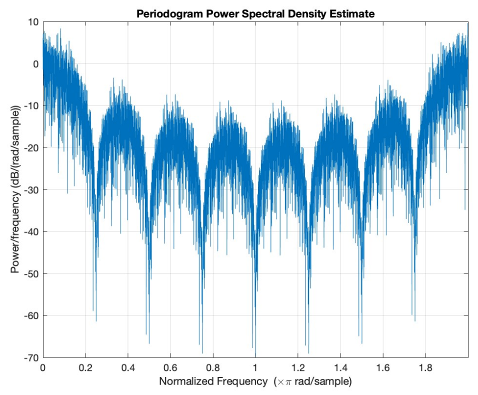


(e) $E_b/N_0 = 8$ dB

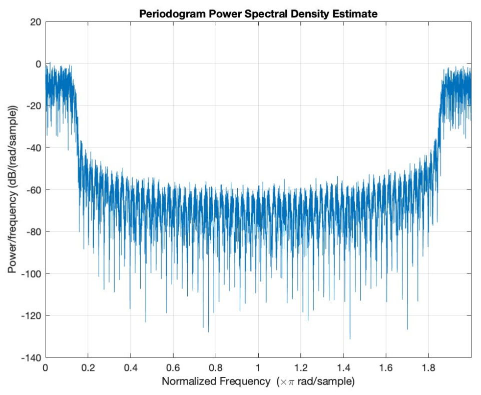


(f) $E_b/N_0 = 10$ dB

Fig. 2: Signal constellation for various values of E_b/N_0.



(a) Rectangular pulse

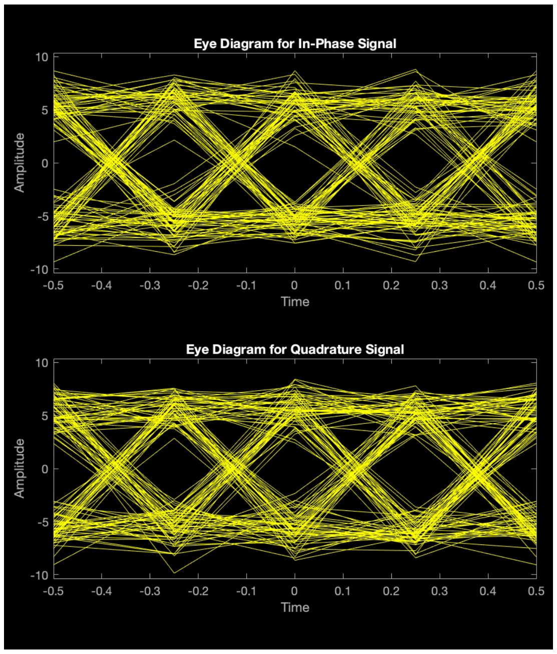


(b) Root raised cosine pulse

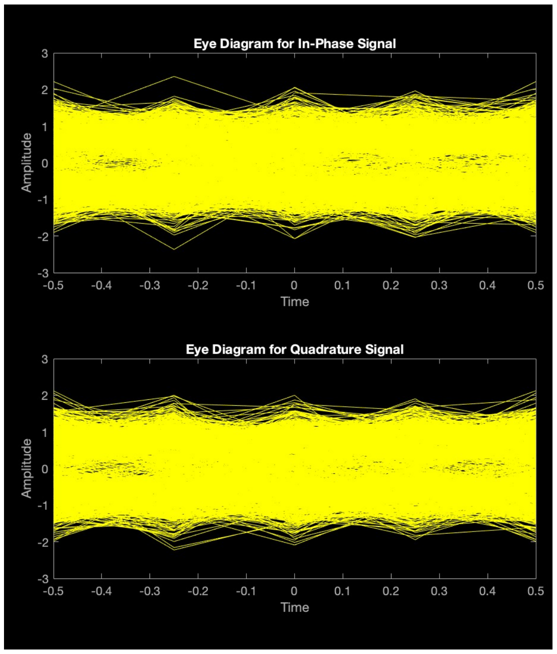
Fig. 3: Power spectral densities for different pulse shapes.

Using the 'multipath' function in MATLAB, a simulation of a multipath channel environment was created. The signal was subjected to the same receiver processes as in an AWGN channel, and the resulting eye diagrams were plotted. As shown in Figure 4, the eye diagram for the two-path ISI channel exhibited significantly more distortion and a smaller eye than the AWGN channel due to the multipath channel's contribution to more ISI.

Figures 5 and 6 depict plots for the BER, synchronization error, and phase estimation error for the



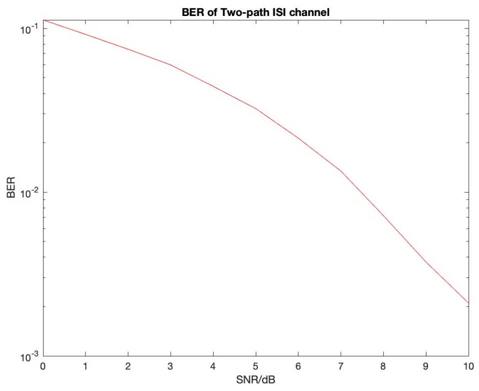
(a) Additive white gaussian noise



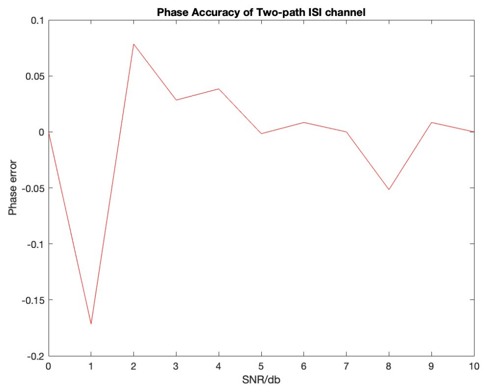
(b) Two-path ISI

Fig. 4: The eye diagram for the AWGN channel and the Multipath channel.

two-path ISI channel at different SNR values. It was observed that the phase estimation accuracy was poor for the two-path ISI channel, but the synchronization error was 0 for a reasonable length of the training sequence. However, the synchronization error became prevalent when the training sequence size was reduced to 8.



(a) BER of Two-path ISI

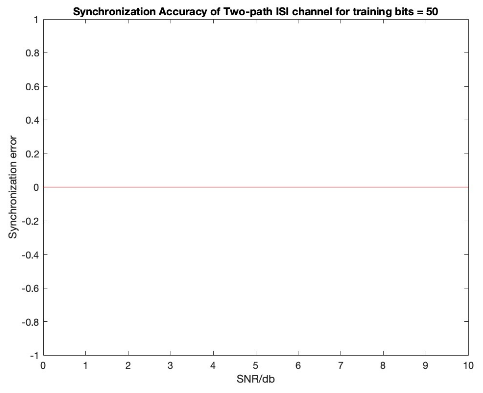


(b) Phase synchronization error for Two-path ISI

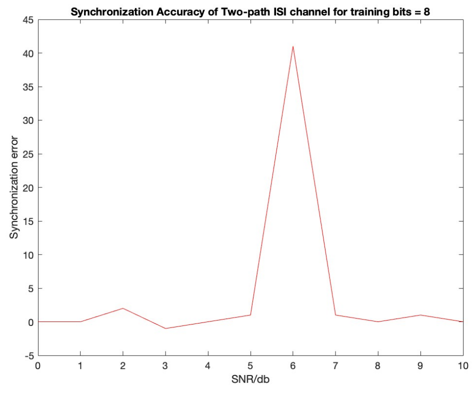
Fig. 5: The implications of ISI channel on the BER and the phase synchronization error for training sequence of length 100

5) *Estimations based on the length of the training sequence*

The simulation examined the performance of phase estimation and synchronization for a training sequence sized $2 - 100$ bits. Furthermore, we examined each performance for E_b/N_0 values from 0 to 20 dB. Once all the values for E_b/N_0 are examined, one can conclude that anything higher than 10 dB can be found reasonable. Hence, the graphs in Figure 7 examine the correlation of the phase error and synchronization error with the training sequence length for a picked SNR of 10 dB. One can also conclude that a length of more than 10 bits for the training sequence may be

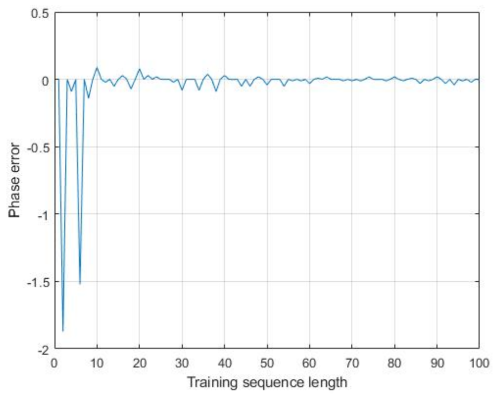


(a) Sequence length of 50

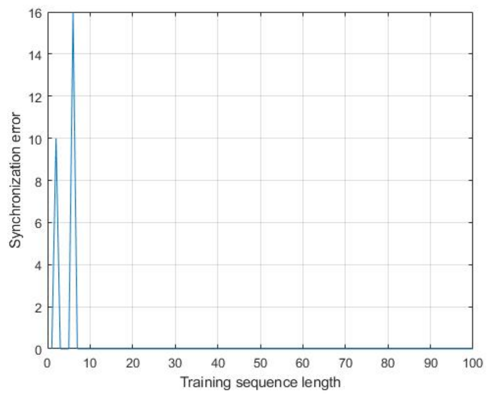


(b) Sequence length of 8

Fig. 6: The implication of ISI channel on the Synchronization error for different sequence lengths



(a) Phase estimation error



(b) Synchronization error

Fig. 7: Performance of phase estimation and synchronization for training sequence length of 100 bits, given $E_b/N_0 = 10$ dB.

considered redundant, as it no longer improves both errors. As a result, the transmission overhead can be computed as: $\frac{\text{Redundant bits}}{\text{Total number of transmitted bits}} = \frac{90}{1000} = 0.09$

V. Conclusions

In conclusion, our project involved simulating a QPSK system. We compared theoretical and simulated results for BER at various Eb/No values. We found that raised root cosine pulse shaping outperforms the rectangular pulse. Higher SNR values improved the constellation diagram, while lower SNR values led to distortion. Investigating the two-path ISI channel highlighted distortions in the eye diagram, causing phase estimation and synchronization errors, especially with shorter training sequences. Through experimentation, we identified practical training sequence lengths for specific Eb/No values. Overall, this project provided practical insights, reinforcing our ability to apply theoretical knowledge.

VI. Appendix

A. *Missing functions*

1) ***Code for the qpsk() function***

```
function d = qpsk(b)
```

```
d=zeros(1,length(b)/2);
%definition of the QPSK symbols using Gray coding.
for n=1:length(b)/2
    p=b(2*n);
    imp=b(2*n-1);
    if (imp==0)&(p==0)
        d(n)=exp(j*pi/4);%45 degrees
    end
    if (imp==1)&(p==0)
        d(n)=exp(j*3*pi/4);%135 degrees
    end
    if (imp==1)&(p==1)
        d(n)=exp(j*5*pi/4);%225 degrees
    end
    if (imp==0)&(p==1)
        d(n)=exp(j*7*pi/4);%315 degrees
    end
end
```

*2) **Code for the sync() function***

```
function t_samp = sync(mf, b_train, Q, t_start, t_end)

% function t_samp = sync(mf, b_train, Q, t_start, t_end)
    qpsk_train = qpsk(b_train);
    len = length(qpsk_train);

%     Initialize the cross-correlation results array
    r = zeros(1, t_end - t_start + 1);

%     Calculate cross-correlation for each time shift
    for shift = t_start:t_end
%         Extract the matched filter output for the corresponding time shift
        a = mf(shift + (0:len-1) * Q);

%         Calculate the absolute value of the inner product
        r(shift - t_start + 1) = abs(sum(conj(a) .* qpsk_train));
    end

%     Find the index of the maximum cross-correlation value
    [~, t_samp] = max(r);
    t_samp = t_samp + t_start - 1;
end
```

*3) **Code for the phase estimation() function***

```
    function phihat = phase_estimation(r, b_train)

phihat = 0;
qpsk_train = qpsk(b_train);
r_train = r(1:length(qpsk_train));
min = norm(r_train - qpsk_train); %minimizing the norm
for phase = -pi:0.01:pi
    r_phi = r_train * exp(-1j*phase);
    min_length = norm(r_phi - qpsk_train);
    if min_length < min
        min = min_length;
        phihat = phase;
    end
end
```

4) Code for the detect() function

```
    function bhat = detect(r)
% bhat = detect(r)

n = length (r);
bhat = zeros(1,2*n);
for i=1:n
    if real(r(i)) > 0
        bhat(2*i-1) = 0;
    else
        bhat(2*i-1) = 1;
    end
    if imag(r(i)) > 0;
        bhat(2*i) = 0;
    else
        bhat(2*i)= 1;
    end
end
```

B. Rest of the code

1) Code for tasks 1 and 3

```
    % Skeleton code for simulation chain
% Simulation for tasks 1 and 3
%(BER performance & Signal Contellation)

% History:
%   2000-06-28  written /Stefan Parkvall
%   2001-10-22  modified /George Jongren

clear

% Initialization
EbN0_db = 0:10;
nr_bits_per_symbol = 2;
nr_guard_bits = 10;




nr_data_bits = 1000;
nr_training_bits = 100;
nr_blocks = 50;
Q = 8;
```

```
% Define the pulse-shape used in the transmitter.
% Pick one of the pulse shapes below or experiemnt
% with a pulse of your own.
pulse_shape = ones(1, Q);
%pulse_shape = root_raised_cosine(Q);

% Matched filter impulse response.
mf_pulse_shape = fliplr(pulse_shape);


% Loop over different values of Eb/No.
nr_errors = zeros(1, length(EbN0_db));   % Error counter
for snr_point = 1:length(EbN0_db)

  % Loop over several blocks to get sufficient statistics.
  for blk = 1:nr_blocks

    %%%
    %%% Transmitter
    %%%

    % Generate training sequence.
    b_train = training_sequence(nr_training_bits);

    % Generate random source data {0, 1}.
    b_data = random_data(nr_data_bits);

    % Generate guard sequence.
    b_guard = random_data(nr_guard_bits);

    % Multiplex training and data into one sequence.
    b = [b_guard b_train b_data b_guard];

    % Map bits into complex-valued QPSK symbols.
    d = qpsk(b);

    % Upsample the signal, apply pulse shaping.
    tx = upfirdn(d, pulse_shape, Q, 1);

    %%%
    %%% AWGN Channel
    %%%

    % Compute variance of complex noise according to report.
    %Needed to add the code into a new line.
    sigma_sqr = norm(pulse_shape)^2 / nr_bits_per_symbol /

    10^(EbN0_db(snr_point)/10);
    % Create noise vector.
    n = sqrt(sigma_sqr/2)*(randn(size(tx))+j*randn(size(tx)));

    % Received signal.
    rx = tx + n;

    %%%
    %%% Receiver
    %%%

    % Matched filtering.
    mf=conv(mf_pulse_shape,rx);

    % Synchronization. The position and size of the search window
    % is here set arbitrarily. Note that you might need to change these
    % parameters. Use sensible values (hint: plot the correlation
    % function used for syncing)!
    t_start=1+Q*nr_guard_bits/2;
    t_end=t_start+50;
    t_samp = sync(mf, b_train, Q, t_start, t_end);

    % Down sampling. t_samp is the first sample, the remaining samples are all
    % separated by a factor of Q. Only training+data samples are kept.
    r = mf(t_samp:Q:t_samp+Q*(nr_training_bits+nr_data_bits)/2-1);

    % Phase estimation and correction.
    phihat = phase_estimation(r, b_train);
    r = r * exp(-j*phihat);

    % Make decisions. Note that dhat will include training sequence bits
    % as well.
    bhat = detect(r);

    % Count errors. Note that only the data bits and not the training bits
    % are included in the comparison. The last data bits are missing as well
    % since the whole impulse response due to the last symbol is not
    % included in the simulation program above.
    temp=bhat(1+nr_training_bits:nr_training_bits+nr_data_bits) ~= b_data;
    nr_errors(snr_point) = nr_errors(snr_point) + sum(temp);

    % Next block.
  end

  % Next Eb/No value.
end

% Compute the BER.
BER_simulation = nr_errors / nr_data_bits / nr_blocks;

EbN0 = 10.^(EbN0_db/10);
```

```
BER_theory = qfunc(sqrt(2*EbN0));
figure;
plot(EbN0_db,BER_simulation);
hold on;
plot(EbN0_db,BER_theory,'r');
set(gca, 'YScale', 'log');
legend('Simulation','Theory');
xlabel('SNR/dB');
ylabel('BER');
grid on;



%Problem 3 - signal contellation
EbN0_db = 0:10;
for snr_point = 1:length(EbN0_db)
    for blk = 1:nr_blocks

    %%% Transmitter
    %%%

    % Generate training sequence.
    b_train = training_sequence(nr_training_bits);

    % Generate random source data {0, 1}.
    b_data = random_data(nr_data_bits);

    % Generate guard sequence.
    b_guard = random_data(nr_guard_bits);

    % Multiplex training and data into one sequence.
    b = [b_guard b_train b_data b_guard];

    % Map bits into complex-valued QPSK symbols.
    d = qpsk(b);

    % Upsample the signal, apply pulse shaping.
    tx = upfirdn(d, pulse_shape, Q, 1);

    %%%
    %%% AWGN Channel
    %%%

    % Compute variance of complex noise according to report.
    % The code is two separate lines due to needed space.
    sigma_sqr = norm(pulse_shape)^2 / nr_bits_per_symbol /

    10^(EbN0_db(snr_point)/10);

    % Create noise vector.
    n = sqrt(sigma_sqr/2)*(randn(size(tx))+j*randn(size(tx)));

    % Received signal.
    rx = tx + n;

    %%%
    %%% Receiver
    %%%

    % Matched filtering.
    mf=conv(mf_pulse_shape,rx);

    % Synchronization. The position and size of the search window
    % is here set arbitrarily. Note that you might need to change these
    % parameters. Use sensible values (hint: plot the correlation
    % function used for syncing)!
    t_start=1+Q*nr_guard_bits/2;
    t_end=t_start+50;
    t_samp = sync(mf, b_train, Q, t_start, t_end);

    % Down sampling. t_samp is the first sample, the remaining samples are all
    % separated by a factor of Q. Only training+data samples are kept.
    r = mf(t_samp:Q:t_samp+Q*(nr_training_bits+nr_data_bits)/2-1);

    % Phase estimation and correction.
    phihat = phase_estimation(r, b_train);
    r = r * exp(-j*phihat);

    % Make decisions. Note that dhat will include training sequence bits
    % as well.
    bhat = detect(r);

    % Count errors. Note that only the data bits and not the training bits
    % are included in the comparison. The last data bits are missing as well
    % since the whole impulse response due to the last symbol is not
    % included in the simulation program above.
    temp=bhat(1+nr_training_bits:nr_training_bits+nr_data_bits) ~= b_data;
    nr_errors(snr_point) = nr_errors(snr_point) + sum(temp);

    % Next block.
    end
    i = abs(EbN0_db(1)) + EbN0_db(snr_point) +1;
    figure (i);
    scatterplot(r);
    ylabel('Q');
    xlabel('I');

end
```

2) Code for tasks 4 and 5

```
% Skeleton code for simulation chain
% Simulation for task 4 and 5 (PSD and eye diagram)

clear

% Initialization
EbN0_db = 0:10;
nr_bits_per_symbol = 2;
nr_guard_bits = 10;
nr_data_bits = 1000;
nr_training_bits = 100;
nr_blocks = 50;
Q = 8;

% Define the pulse-shape used in the transmitter.

  pulse_shape = ones(1, Q);
% pulse_shape = root_raised_cosine(Q);

% Matched filter impulse response.
mf_pulse_shape = fliplr(pulse_shape);


% Loop over different values of Eb/No.

nr_errors1 = zeros(1,length(EbN0_db));
nr_errors2 = zeros(1,length(EbN0_db));
phihat_r1 = zeros(1,length(EbN0_db));
phihat_isi = zeros(1,length(EbN0_db));
t_psamp1 = zeros(1,length(EbN0_db));
t_psamp2 = zeros(1,length(EbN0_db));


% Loop over several blocks to get sufficient statistics.
for snr_point = 1:length(EbN0_db)
  for blk = 1:nr_blocks

    %%%
    %%% Transmitter
    %%%

    % Generate training sequence.
    b_train = training_sequence(nr_training_bits);

    % Generate random source data {0, 1}.
    b_data = random_data(nr_data_bits);

    % Generate guard sequence.
    b_guard = random_data(nr_guard_bits);
```

```
% Multiplex training and data into one sequence.
b = [b_guard b_train b_data b_guard];

% Map bits into complex-valued QPSK symbols.
d = qpsk(b);

% Upsample the signal, apply pulse shaping.
tx = upfirdn(d, pulse_shape, Q, 1);

%%%
%%% two path isi
%%%

t = 6;
tx2 = multipath(tx,t);

% Compute variance of complex noise
% The code is two separate lines due to needed space.
sigma_sqr2 = norm(pulse_shape)^2 / nr_bits_per_symbol /

10^(EbN0_db(snr_point)/10);

% Create noise vector.
n2 = sqrt(sigma_sqr2/2)*(randn(size(tx2))+1j*randn(size(tx2)));
rx2 = tx2 + n2;


%%%
%%% AWGN Channel
%%%

% Compute variance of complex noise
% The code is two separate lines due to needed space.
sigma_sqr = norm(pulse_shape)^2 / nr_bits_per_symbol /

10^(EbN0_db(snr_point)/10);

% Create noise vector.
n = sqrt(sigma_sqr/2)*(randn(size(tx))+j*randn(size(tx)));

% Received signal.
rx = tx + n;

%%%
%%% Receiver
%%%

% Matched filtering.
mf=conv(mf_pulse_shape,rx);
```

```
    mf2=conv(mf_pulse_shape,rx2);

    % Synchronization

    t_start=1+Q*nr_guard_bits/2;
    t_end=t_start+50;
    t_samp = sync(mf, b_train, Q, t_start, t_end);
    t_samp2 = sync(mf2, b_train, Q, t_start, t_end);
    t_psamp2(snr_point) = t_samp2 - 48;

    % Down sampling

    r = mf(t_samp:Q:t_samp+Q*(nr_training_bits+nr_data_bits)/2-1);
    r2 = mf2(t_samp2:Q:t_samp2+Q*(nr_training_bits+nr_data_bits)/2-1);

    % Phase estimation and correction

    phihat = phase_estimation(r, b_train);
    r = r * exp(-j*phihat);

     phihat2 = phase_estimation(r2, b_train);
    phihat_isi(snr_point) = phihat2;
     r2 = r2 * exp(-1i*phihat2);

    % decisions
    bhat = detect(r);
    bhat2 = detect(r2);

    % Count errors

    temp=bhat(1+nr_training_bits:nr_training_bits+nr_data_bits) ~= b_data;
    nr_errors1(snr_point) = nr_errors1(snr_point) + sum(temp);

    temp2 = bhat2(1+nr_training_bits:nr_training_bits+nr_data_bits) ~= b_data;
    nr_errors2(snr_point) = nr_errors2(snr_point) + sum(temp2);

    % Next block.
  end

  % Next Eb/No value.
end

% Compute the BER.
BER_AWGN = nr_errors1 / nr_data_bits / nr_blocks;
BER_ISI = nr_errors2 / nr_data_bits / nr_blocks;
EbN0 = 10.^(EbN0_db/10);

figure(1);

plot(EbN0_db,BER_ISI,'r');
```

```
set(gca, 'YScale', 'log');
xlabel('SNR/dB');
ylabel('BER');
title('BER of Two-path ISI channel');

figure(2);

plot(EbN0_db,phihat_isi,'r');
title('Phase Accuracy of Two-path ISI channel');
xlabel('SNR/db');
ylabel('Phase error')

figure(3);

plot(EbN0_db,t_psamp2,'r');
title('Synchronization Accuracy of Two-path ISI channel for training bits = 8')
xlabel('SNR/db');
ylabel('Synchronization error');

figure;
periodogram(tx);

figure;
eyediagram(rx,4);
```

3) Code for task 6

```
    % History:
%   2000-06-28  written /Stefan Parkvall
%   2001-10-22  modified /George Jongren
clc;
clear;

% Initialization
EbN0_db =10;
nr_bits_per_symbol = 2;
nr_guard_bits = 10;




nr_data_bits = 1000;
%nr_training_bits = 100;
nr_blocks = 50;
Q = 8;

% Define the pulse-shape used in the transmitter.
% Pick one of the pulse shapes below or experiemnt
% with a pulse of your own.
pulse_shape = ones(1, Q);
%pulse_shape = root_raised_cosine(Q);
```

```
% Matched filter impulse response.
mf_pulse_shape = fliplr(pulse_shape);


% Loop over different values of Eb/No.
nr_errors = zeros(1, 100);   % Error counter
phase = zeros(1,100);
t_psamp = zeros(1,100);
for nr_training_bits = 2:2:100

  % Loop over several blocks to get sufficient statistics.
  for blk = 1:nr_blocks

    %%%
    %%% Transmitter
    %%%

    % Generate training sequence.
    b_train = training_sequence(nr_training_bits);

    % Generate random source data {0, 1}.
    b_data = random_data(nr_data_bits);

    % Generate guard sequence.
    b_guard = random_data(nr_guard_bits);

    % Multiplex training and data into one sequence.
    b = [b_guard b_train b_data b_guard];

    % Map bits into complex-valued QPSK symbols.
    d = qpsk(b);

    % Upsample the signal, apply pulse shaping.
    tx = upfirdn(d, pulse_shape, Q, 1);

    %%%
    %%% AWGN Channel
    %%%

    % Compute variance of complex noise according to report.
    sigma_sqr = norm(pulse_shape)^2 / nr_bits_per_symbol / 10^(EbN0_db/10);

    % Create noise vector.
    n = sqrt(sigma_sqr/2)*(randn(size(tx))+1j*randn(size(tx)));

    % Received signal.
    rx = tx + n;

    %%%
    %%% Receiver
    %%%

    % Matched filtering.
    mf=conv(mf_pulse_shape,rx);

    % Synchronization. The position and size of the search window
    % is here set arbitrarily. Note that you might need to change these
    % parameters. Use sensible values (hint: plot the correlation
    % function used for syncing)!
    t_start=1+Q*nr_guard_bits/2;
    t_end=t_start+50;
    t_samp = sync(mf, b_train, Q, t_start, t_end);
    t_psamp(nr_training_bits) = t_samp-48;

    % Down sampling. t_samp is the first sample, the remaining samples are all
    % separated by a factor of Q. Only training+data samples are kept.
    r = mf(t_samp:Q:t_samp+Q*(nr_training_bits+nr_data_bits)/2-1);

    % Phase estimation and correction.
    phihat = phase_estimation(r, b_train);
    phase(nr_training_bits) = phihat;
    r = r * exp(-1i*phihat);

    % Make decisions. Note that dhat will include training sequence bits
    % as well.
    bhat = detect(r);

    % Count errors. Note that only the data bits and not the training bits
    % are included in the comparison. The last data bits are missing as well
    % since the whole impulse response due to the last symbol is not
    % included in the simulation program above.
    temp=bhat(1 +nr_training_bits:nr_training_bits+nr_data_bits) ~= b_data;
    nr_errors(nr_training_bits) = nr_errors(nr_training_bits) + sum(temp);

    % Next block.
  end

  % Next Eb/No value.
end
%     figure (phase);
%     figure (t_psamp);
%     scatterplot(phase);
%     scatterplot(t_psamp);
%     yline(0)
%     xline(0)

% Compute the BER.
BER = nr_errors / nr_data_bits / nr_blocks;

figure(1);
```

```
plot(t_psamp);
%title('Synchronization error vs. Training sequence length');
xlabel('Training sequence length');
ylabel('Synchronization error');
grid on;

figure(2);
plot(phase);
%title('Phase error vs. Training sequence length');
xlabel('Training sequence length');
ylabel('Phase error');
grid on;
```

VII. References

1. Fundamentals of Digital Communication, Upamanyu Madhow. Cambridge University Press.