EE 6390-Introduction to Wireless Communications Systems

Spring-2015

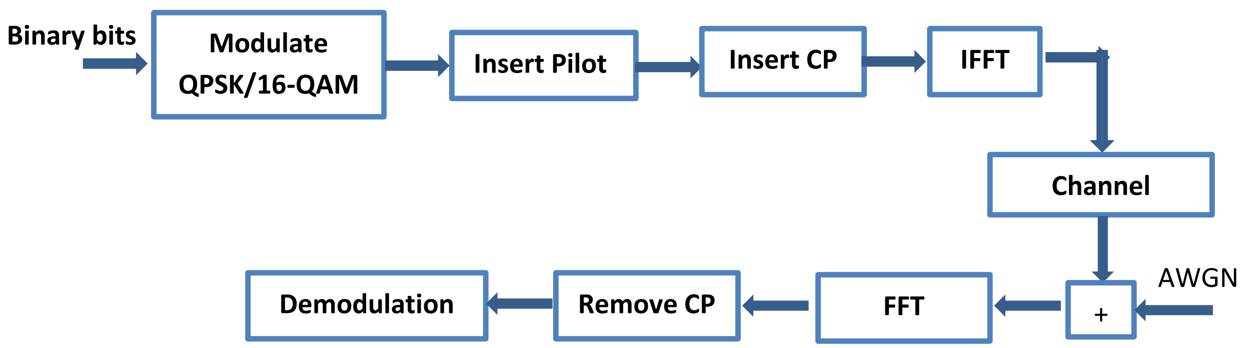
Final Project Report

Physical Layer Simulation of a Simplified LTE-OFDM System

Tasmiah S. Khan (ID: 2021200236)
Gayathri N. Kannepalli (ID: 2021229050)

Project Description

This project entails simulation of simplified LTE-OFDM system in Matlab. The modulation schemes QPSK and 16-QAM are used to modulate the data. First binary data streams is generated and modulated using the two modulation schemes. Four pilots and cyclic prefix are then added. The modulated signals are transmitted over AWGN channel as well as Rayleigh fading multipath channel. The Bit Error Rate (BER) are computed under each scenario. Finally, the empirical spectral efficiency is studied for adaptive modulation scheme. A high level block diagram of the system is shown in the figure below.



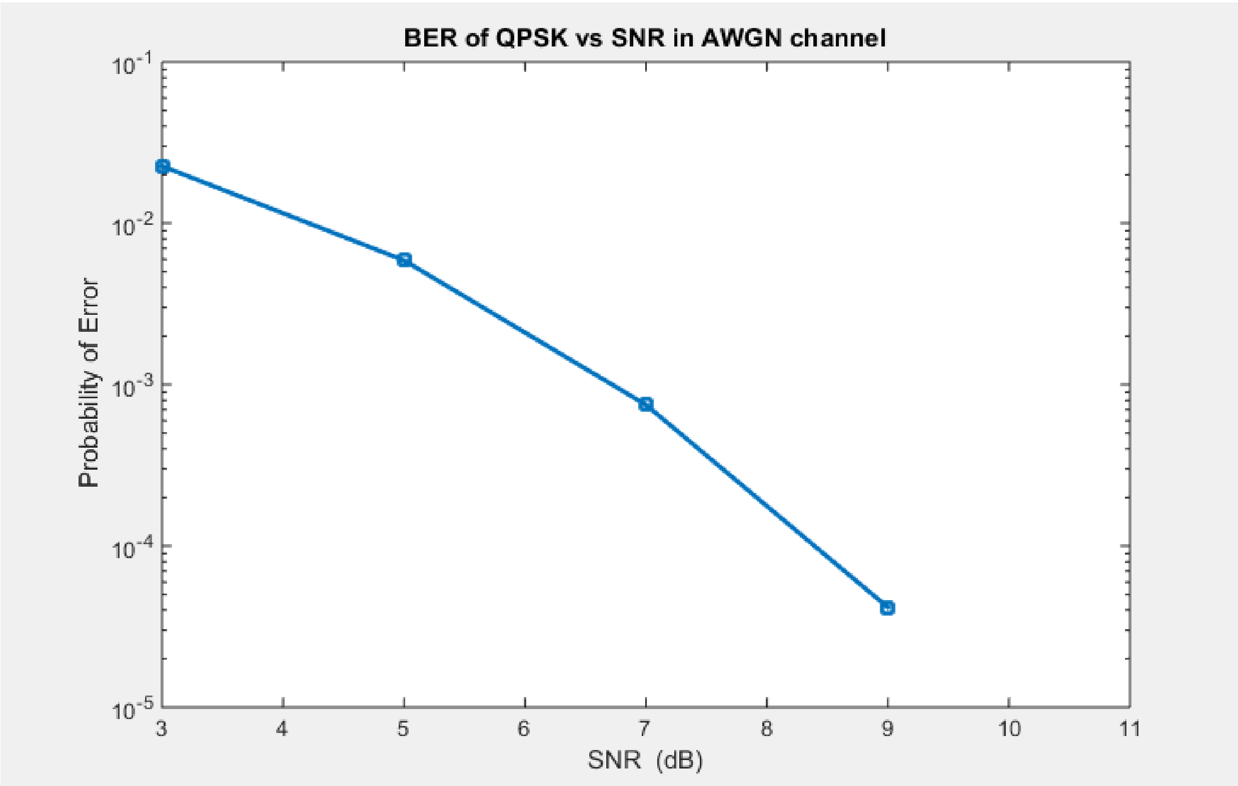
The project work was equally divided between both of us. One of us worked on Adaptive modulation and BER calculation with AWGN channel and the other worked on the BER calculation with multipath channel as well as integrating all the code to make the simulation work. The contribution on the project report was equal.

System Parameters

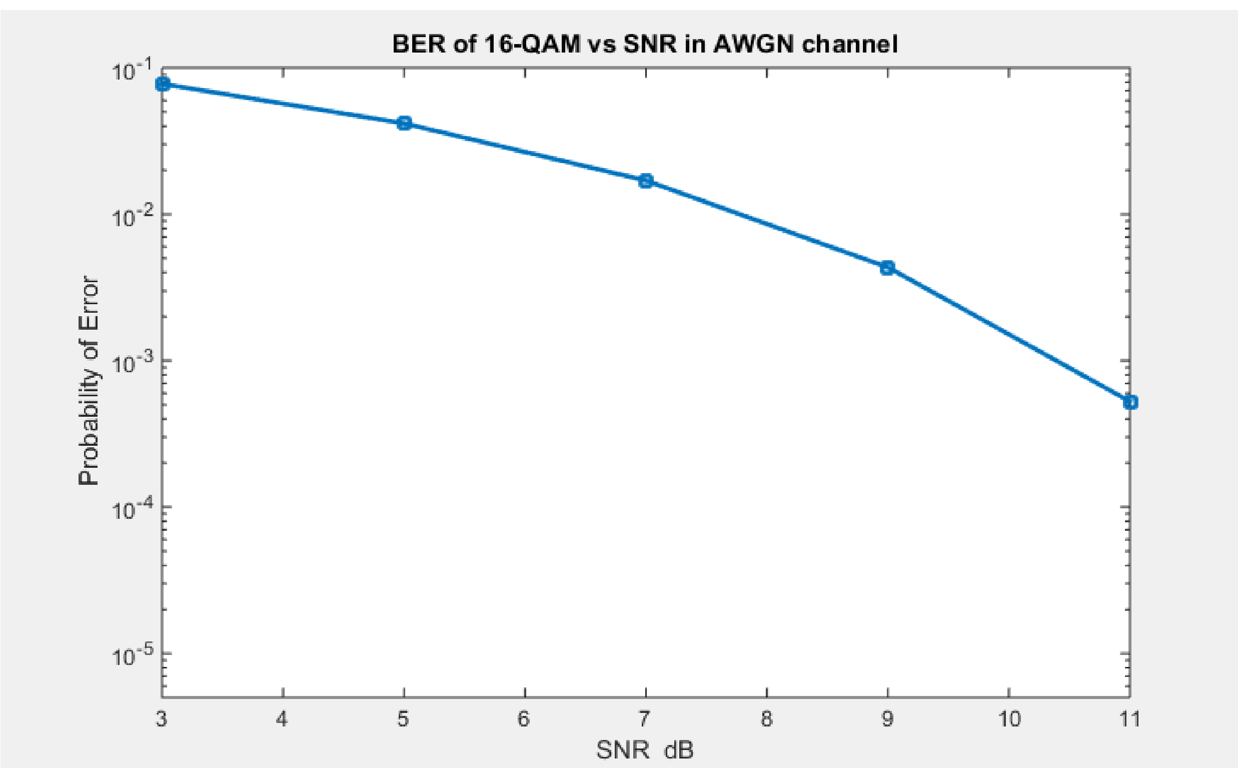
The following parameters were used to simulate the system

Channel Bandwidth = 10MHz
Modulation Types: QPSK, 16-QAM
FFT Size = 64
CP Length = 16
No. of used subcarriers (N_{used}) = 52
No. of pilot subcarriers (N_{ref}) = 4
No. of data subcarriers per symbol (N_{data}) = 48
No. of null subcarriers ($N_{left}, N_{dc}, N_{right}$) = 12

The following plots of BER are generated for QPSK and 16-QAM under AWGN channel

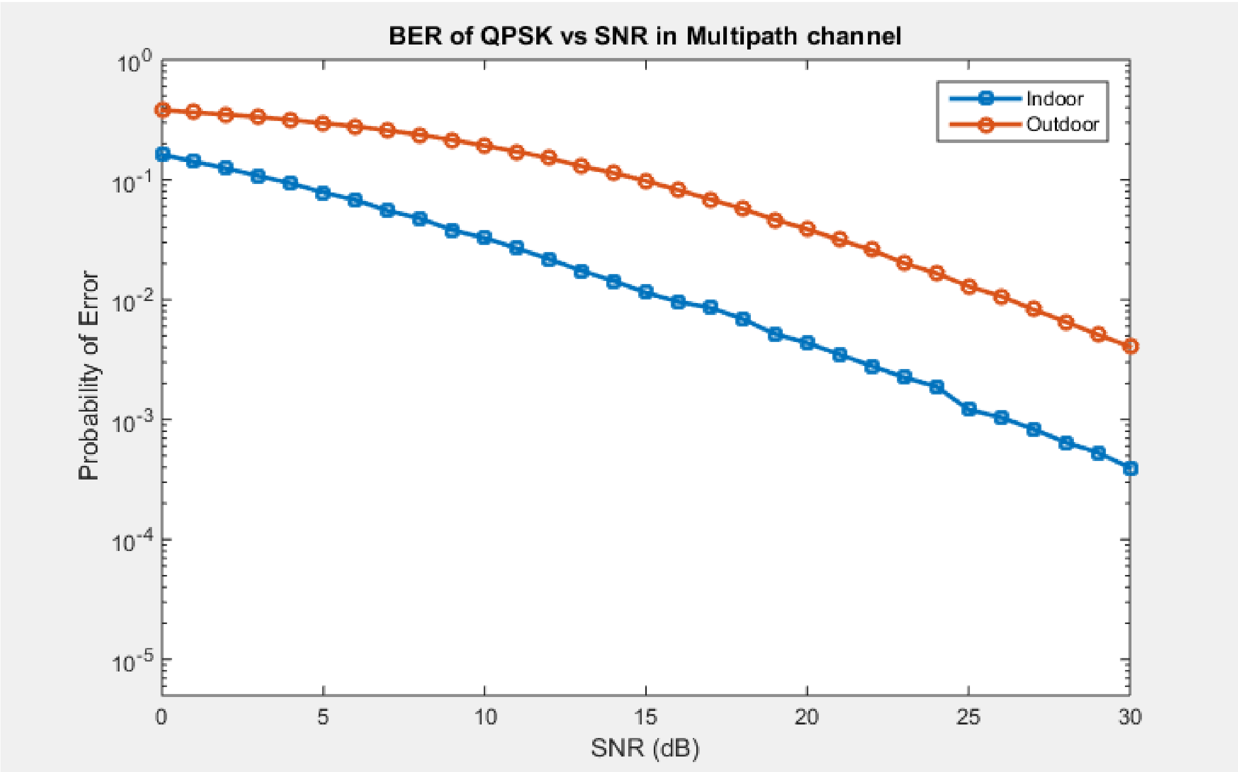


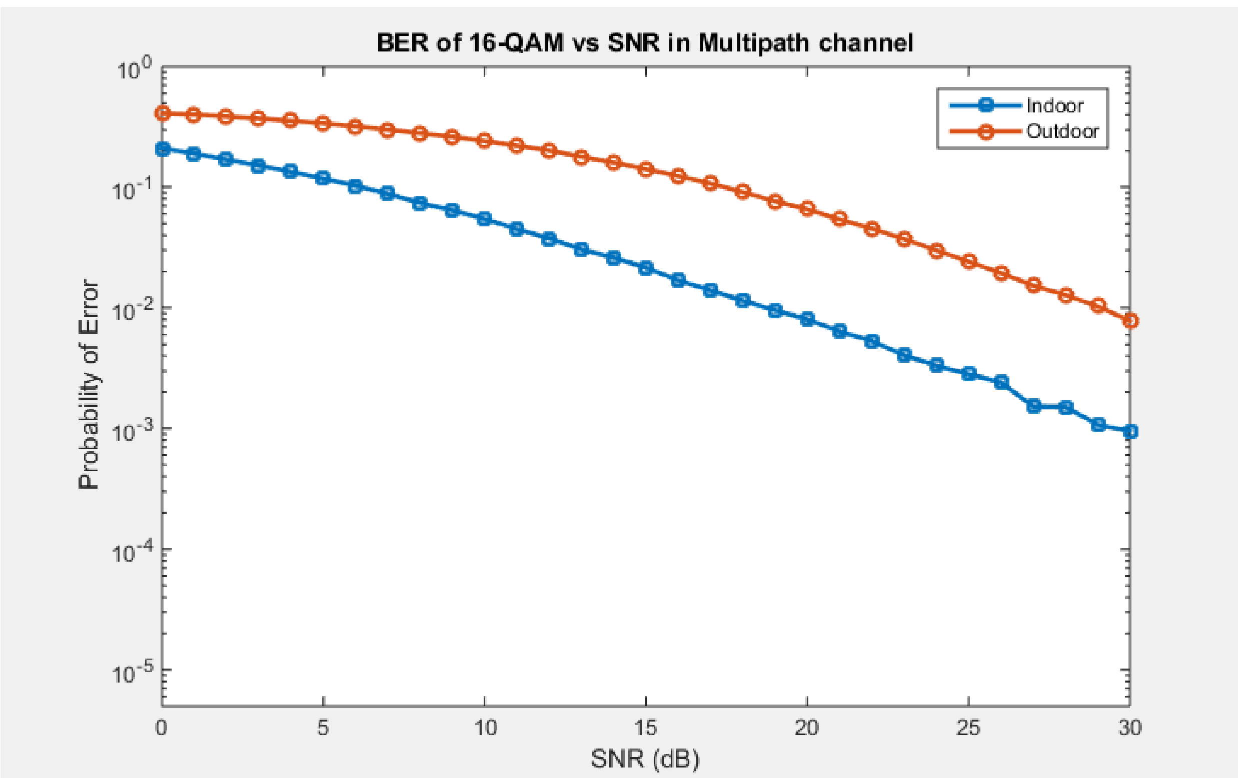
BER of QPSK vs SNR in AWGN channel
Probability of Error
SNR (dB)



BER of 16-QAM vs SNR in AWGN channel
Probability of Error
SNR dB

For Multipath Channel simulation, Channel B was used. BER was calculated for indoor and outdoor models for both QPSK and 16-QAM. The plots are shown below:





The plot of empirical spectral efficiency for the indoor and outdoor channel models using adaptive modulation is shown below:

Matlab Code

```
function LTE_OFDM_SIMULATION()
clc
close all;
clear all;

sequence = rand(1,4)>0.5;  %pseudo binary sequence
pilot =  2.*sequence - 1; %Pilot signal
range = 10:30;  %Adaptive Modulation range
S1 = 100; %100 symbols
S2 = 10; %The number of Monte-Carlo trials for each symbol
AWGN_SNR_dB = [3 5 7 9 11];
AWGN_SNR = 10.^((AWGN_SNR_dB)./10); %SNR in linear scale
        for i = 1:S1
            inputQPSK = rand(1,2*48)>0.5;  %input to QPSK modulator
            inputQAM = rand(1,4*48)>0.5;   %input to 16 QAM modulator

            for j = 1:length(AWGN_SNR)
                for k = 1:S2
%QPSK Modulation
QPSK_Modulated = QPSK(1,inputQPSK); %QPSK Modulation
QPSK_OFDM = Transmitter_OFDM(QPSK_Modulated,pilot);  %OFDM Transmission
QPSK_OFDM_Channel = AWGN(QPSK_OFDM,AWGN_SNR(j));  %AWGN Channel
[QPSKDemodulated_OFDM var] = Receiver_OFDM(1,QPSK_OFDM_Channel,[],[],[]); %OFDM Receiver
QPSK_Demodulated = QPSK(2,QPSKDemodulated_OFDM);   %QPSK Demodulation
error1(k) = sum(inputQPSK ~= QPSK_Demodulated)/(48*2);%bit error probability
%16 QAM Modulation
QAM_Modulated = QAM(1,inputQAM); %QAM Modulation
QAM_OFDM = Transmitter_OFDM(QAM_Modulated,pilot);  %OFDM Transmission
QAM_OFDM_Channel = AWGN(QAM_OFDM,AWGN_SNR(j));  %AWGN Channel
[QAMDemodulated_OFDM var] = Receiver_OFDM(1,QAM_OFDM_Channel,[],[],[]); %OFDM Receiver
QAM_Demodulated = QAM(2,QAMDemodulated_OFDM);   %QAM Demodulation
error2(k) = sum(inputQAM ~= QAM_Demodulated)/(48*4); %bit error probability

                end
                sym_error1(i,j) = sum(error1)/S2;
                sym_error2(i,j) = sum(error2)/S2;
            end

        end

        for j = 1:length(AWGN_SNR)
            bit_error1(j) = sum(sym_error1(:,j))/S1;
            ber_error2(j) = sum(sym_error2(:,j))/S1;
        end

                figure
                semilogy(AWGN_SNR_dB,bit_error1);
                title('BER of QPSK vs SNR in AWGN channel');
                xlabel('SNR  (dB)');
                ylabel('Probability of Error');

                figure
                semilogy(AWGN_SNR_dB,ber_error2);
                title('BER of 16-QAM vs SNR in AWGN channel');
                xlabel('SNR  dB');
```

```
                ylabel('Probability of Error');

        AWGN_SNR_dB = 0:30;
        AWGN_SNR = 10.^((AWGN_SNR_dB)./10); %SNR in linear scale
        for i = 1:S1

            inputQPSK = rand(1,2*48)>0.5;  %input to QPSK modulator
            inputQAM = rand(1,4*48)>0.5;   %input to 16 QAM modulator
            [tapsIndoor tapsOutdoor] = Multipath_Taps();

            for j = 1:length(AWGN_SNR)
                for k = 1:S2
                %QPSK Modulation
                 QPSK_Modulated = QPSK(1,inputQPSK); %QPSK Modulation
                 QPSK_OFDM = Transmitter_OFDM(QPSK_Modulated,pilot);  %OFDM Transmission
                 [QPSK_Indoor_OFDM_Channel QPSK_Outdoor_OFDM_Channel] = Channel_multipath(QPSK_OFDM
, AWGN_SNR(j), tapsIndoor, tapsOutdoor);  %Multipath Channel
                 [QPSKDemodulated_Indoor_OFDM QPSKDemodulated_Outdoor_OFDM] = Receiver_OFDM(2,QPSK_
Indoor_OFDM_Channel,QPSK_Outdoor_OFDM_Channel,tapsIndoor,tapsOutdoor); %OFDM Receiver
                 QPSKDemodulated_Indoor = QPSK(2,QPSKDemodulated_Indoor_OFDM);   %QPSK Demodulation
 for Indoor
                 QPSKDemodulated_Outdoor = QPSK(2,QPSKDemodulated_Outdoor_OFDM);   %QPSK Demodulati
on for Outdoor
                 error_Indoor1(k) = sum(inputQPSK ~= QPSKDemodulated_Indoor)/(48*2);   %BER for Ind
oor
                 error_Outdoor1(k) = sum(inputQPSK ~= QPSKDemodulated_Outdoor)/(48*2);   %BER for O
utdoor
                 %16 QAM Modulation
                 QAM_Modulated = QAM(1,inputQAM); %QAM Modulation
                 QAM_OFDM = Transmitter_OFDM(QAM_Modulated,pilot);  %OFDM Transmission
                 [QAM_Indoor_OFDM_Channel QAM_Outdoor_OFDM_Channel] = Channel_multipath(QAM_OFDM, A
WGN_SNR(j), tapsIndoor, tapsOutdoor);  %Multipath Channel
                 [QAMDemodulated_Indoor_OFDM QAM_Demod_OFDMP] = Receiver_OFDM(2,QAM_Indoor_OFDM_Cha
nnel,QAM_Outdoor_OFDM_Channel,tapsIndoor,tapsOutdoor); %OFDM Receiver
                 QAMDemodulated_Indoor = QAM(2,QAMDemodulated_Indoor_OFDM);   %QAM Demodulation for
 Indoor
                 QAMDemodulated_Outdoor = QAM(2,QAM_Demod_OFDMP);   %QAM Demodulation for Outdoor
                 error_Indoor2(k) = sum(inputQAM ~= QAMDemodulated_Indoor)/(48*4);   %BER for Indoo
r
                 error_Outdoor2(k) = sum(inputQAM ~= QAMDemodulated_Outdoor)/(48*4);   %BER for Out
door
                end
                per_Indoor1(i,j) = sum(error_Indoor1)/S2;
                per_Outdoor1(i,j) = sum(error_Outdoor1)/S2;
                per_Indoor2(i,j) = sum(error_Indoor2)/S2;
                per_Outdoor2(i,j) = sum(error_Outdoor2)/S2;
            end

        end

        for j = 1:length(AWGN_SNR)   %Total Probability
            ber_Indoor1(j) = sum(per_Indoor1(:,j))/S1;
            ber_Outdoor1(j) = sum(per_Outdoor1(:,j))/S1;
            ber_Indoor2(j) = sum(per_Indoor2(:,j))/S1;
            ber_Outdoor2(j) = sum(per_Outdoor2(:,j))/S1;
        end
                %QPSK
```

```
                figure
                semilogy(AWGN_SNR_dB,ber_Indoor1);
                hold on;
                semilogy(AWGN_SNR_dB,ber_Outdoor1);
                legend('Indoor','Outdoor',1);
                title('BER of QPSK vs SNR in Multipath channel');
                xlabel('SNR (dB)');
                ylabel('Probability of Error');

                %16 QAM
                figure
                semilogy(AWGN_SNR_dB,ber_Indoor2);
                hold on;
                semilogy(AWGN_SNR_dB,ber_Outdoor2);
                legend('Indoor','Outdoor',1);
                title('BER of 16-QAM vs SNR in Multipath channel');
                xlabel('SNR (dB)');
                ylabel('Probability of Error');

 %Adaptive Modulation
    Indoor_SpectralEff = zeros(1,length(range));
    Outdoor_SpectralEff = zeros(1,length(range));
    for k = 1:S1
        [tapsIndoor tapsOutdoor] = Multipath_Taps();
        [spectralEff_Indoor spectralEff_Outdoor] = adaptive_Modulation(tapsIndoor,tapsOutdoor);
        Indoor_SpectralEff = Indoor_SpectralEff + spectralEff_Indoor;
        Outdoor_SpectralEff = Outdoor_SpectralEff + spectralEff_Outdoor;
    end

    Indoor_SpectralEff = Indoor_SpectralEff./S1;
    Outdoor_SpectralEff = Outdoor_SpectralEff./S1;

    figure
    plot(range,Indoor_SpectralEff);
    hold on;
    plot(range,Outdoor_SpectralEff);
    legend('Indoor','Outdoor',2);
    title('Spectral Efficiency vs SNR');
    xlabel('SNR (dB)');
    ylabel('Spectral Efficiency');

end
% Functions:
% QPSK Modulation:
function [output] = QPSK(value, input)
QPSK_signals = [-1-1i -1+1i 1+1i 1-1i]; %corresponding to [-3pi/4 3pi/4 pi/4, -pi/4]
if (value == 1)
    QPSK_bits = 2.*input - 1;  %mapping the bits to +1 or -1
    seperation = reshape(QPSK_bits,2,length(input)/2); %QPSK in-phase/quadrature-phase seperation
    QPSK_inphase = seperation(1,:);  %in-phase component (input) for QPSK
    QPSK_quadrature = seperation(2,:);  %quadrature-phase component (input) for QPSK
    output = QPSK_inphase + (1i.*QPSK_quadrature);
else
    output = zeros(1,length(input)*2);
    QPSK_distance = [abs(input - QPSK_signals(1));
              abs(input - QPSK_signals(2));
              abs(input - QPSK_signals(3));
```

```
                abs(input - QPSK_signals(4))];

    for m = 1:length(input)
        QPSK_symbol = QPSK_signals(find(QPSK_distance(:,m)==min(QPSK_distance(:,m)),1,'first'));
        output(2*m - 1) = (real(QPSK_symbol)+1)/2;
        output(2*m) = (imag(QPSK_symbol)+1)/2;
    end
end
end
%16-QAM Modulation:
function [output] = QAM(value, input)
QAM_symbols = (2/sqrt(10)).*[-3 -1 3 1]; %corresponding to [00 01 10 11] and making Eb = 1
if (value == 1)
    QAM_seperation = reshape(input,4,length(input)/4);  %QAM in-phase/quadrature-phase bits seperat
ion
    d1 = QAM_seperation(1,:);  %d1 of QAM
    d2 = QAM_seperation(2,:);  %d2 of QAM
    d3 = QAM_seperation(3,:);  %d3 of QAM
    d4 = QAM_seperation(4,:);  %d4 of QAM
    QAM_inphase = zeros(1,length(input)/4);
    QAM_seperation = zeros(1,length(input)/4);

    for i = 1:length(input)/4
        QAM_inphase(i) = QAM_symbols(((2*d1(i))+d2(i))+ 1);
        QAM_seperation(i) = QAM_symbols(((2*d3(i))+d4(i))+ 1);
    end
output = QAM_inphase + (1i.*QAM_seperation);
else
    output = zeros(1,length(input)*4);
    QAM_distance = zeros(length(QAM_symbols)*4,length(input));
    for i = 1:(length(QAM_symbols)*4)
        bin = dec2bin(i-1,4);
        QAM_distance(i,:) = abs(input - ((QAM_symbols(bin2dec(bin(1:2))+ 1))+(1i*(QAM_symbols(bin2d
ec(bin(3:4))+ 1)))));
    end

    for i = 1:length(input)
        bin = dec2bin((find(QAM_distance(:,i)==min(QAM_distance(:,i)),1,'first')-1),4);
        output(4*i - 3) = str2double(bin(1));
        output(4*i - 2) = str2double(bin(2));
        output(4*i - 1) = str2double(bin(3));
        output(4*i) =  str2double(bin(4));
    end
end
end
% AWGN Channel:
function [signal_received] = AWGN(input, awgnSNR)
inputLength = length(input);   %Input length
signal_inphase = real(input) + (1/sqrt(2*awgnSNR))*randn(1,inputLength); % In-phase received signal
signal_quadrature = imag(input) + (1/sqrt(2*awgnSNR))*randn(1,inputLength);% Quadrature received si
gnal
signal_received = signal_inphase + (1i.*signal_quadrature);
end
%Multipath Channel Taps:
function [tapsIndoor tapsOutdoor] = Multipath_Taps()
sampleDuration = 1/(10*10^6);   %Channel bandwidth = 10MHz
delayIndoor = (10^-9).*[0 100 200 300 500 700]; %Delay profile for Indoor channel model
```

```
relativePower_IndoordB = [0 -3.6 -7.2 -10.8 -18 -25.2]; %Power profile for indoor channel model
delayOutdoor = (10^-9).*[0 5 30 45 75 90 105 140 210 230 250 270 275 475 595 690]; %Delay profile f
or Outdoor Multipath channel model
relativePower_OutdoordB = [-1.5 -10.2 -16.6 -19.2 -20.9 -20.6 -16.6 -16.6 -23.9 -12 -23.9 -21 -17.7
 -24.6 -22 -29.2]; %Power profile for Outdoor Multipath channel model

tapsIndoor = zeros(1,80);
tapsOutdoor = zeros(1,80);

relativePower_Indoor = 10.^(relativePower_IndoordB./10);
relativePower_Outdoor = 10.^(relativePower_OutdoordB./10);

for i = 1:length(delayIndoor)
    tapsIndoor(round(delayIndoor(i)/sampleDuration) + 1) = sqrt(relativePower_Indoor(i)*((randn(1,1
)^2) + (randn(1,1)^2))/2);
end
for m = 1:length(delayOutdoor)
tapsOutdoor(round(delayOutdoor(m)/sampleDuration) + 1) = sqrt(relativePower_Outdoor(m)*((randn(1,1)
^2) + (randn(1,1)^2))/2);
end
end
% Multipath Channels:
function [indoorChan outdoorChan] = Channel_multipath(input, awgnSNR, tapsIndoor, tapsOutdoor)
inputLength = length(input);   %Input length
noise_inphase = (1/sqrt(2*awgnSNR))*randn(1,inputLength);    %In-phase noise
noise_quadrature = (1/sqrt(2*awgnSNR))*randn(1,inputLength);    %Quadrature-phase noise

indoorChan = ifft((fft(input,inputLength).*fft(tapsIndoor,inputLength)),inputLength) + (noise_inpha
se + (1i*noise_quadrature));
outdoorChan = ifft((fft(input,inputLength).*fft(tapsOutdoor,inputLength)),inputLength) + (noise_inp
hase + (1i*noise_quadrature));
end
% OFDM Transmit:
function [output] = Transmitter_OFDM(input, pilot)
fft_size = 64;  %Length of fft/ifft
Length_cp = 16; %Length of cyclic prefix
Begin_cp = fft_size - Length_cp + 1;
End_cp = fft_size;

ofdmSequence = [zeros(1,6) input(1:5) pilot(1) input(6:18) pilot(2) input(19:24) zeros(1,1) input(2
5:30) pilot(3) input(31:43) pilot(4) input(44:48) zeros(1,5)];

inputIFFT = sqrt(fft_size)*ifft(ofdmSequence,64);  %IFFT
output = [inputIFFT(Begin_cp:End_cp) inputIFFT];   %Adding Cyclic Prefix
end
% OFDM Receive:
function [IndoorDetection OutdoorDetection] = Receiver_OFDM(value,indoor, outdoor, IndoorTaps, Outd
oorTaps)
fft_size = 64;   %Length of fft/ifft
outputLength = length(indoor);   %Length of received signal (same for both Indoor and Outdoor)

if (value == 1)  %value = 1 for AWGN channel
    Indoorfft = (1/sqrt(fft_size))*fft(indoor(17:80),fft_size);
    IndoorDetection = [Indoorfft(7:11) Indoorfft(13:25) Indoorfft(27:32) Indoorfft(34:39) Indoorfft
(41:53) Indoorfft(55:59)];
    OutdoorDetection = [];
else
```

```
indoor = ifft((fft(indoor,outputLength)./fft(IndoorTaps,outputLength)),outputLength);
outdoor = ifft((fft(outdoor,outputLength)./fft(OutdoorTaps,outputLength)),outputLength);
Indoorfft = (1/sqrt(fft_size))*fft(indoor(17:80),fft_size);
Outdoorfft = (1/sqrt(fft_size))*fft(outdoor(17:80),fft_size);

IndoorDetection = [Indoorfft(7:11) Indoorfft(13:25) Indoorfft(27:32) Indoorfft(34:39) Indoorfft(41:
53) Indoorfft(55:59)];
OutdoorDetection = [Outdoorfft(7:11) Outdoorfft(13:25) Outdoorfft(27:32) Outdoorfft(34:39) Outdoorf
ft(41:53) Outdoorfft(55:59)];
end
end
% Adaptive modulation:
function [spectralEffIndoor spectralEffOutdoor] = adaptive_Modulation(tapsIndoor,tapsOutdoor)
lengh = 64;
SNRrange = 10:30; %range of average SNR is from 10 to 30dB
noisePower = 1./(10.^((SNRrange)./10));

    fftIndoor = fft(tapsIndoor,lengh);
    fftOutdoor = fft(tapsOutdoor,lengh);

spectralEffOutdoor = zeros(1,length(SNRrange));
spectralEffIndoor = zeros(1,length(SNRrange));

for i = 1:length(SNRrange)
    SNRIndoor = 10*log10((abs(fftIndoor).^2)./noisePower(i));
    SNROutdoor = 10*log10((abs(fftOutdoor).^2)./noisePower(i));
    nbits = 0;
    for j = 1:length(SNRIndoor)
         if ((SNRIndoor(j) >= 10.61) && (SNRIndoor(j) < 13.94))   %QPSK
                nbits = nbits + 2;
                 else if ((SNRIndoor(j) >= 17.25) && (SNRIndoor(j) < 20.4))   %16 QAM
                        nbits = nbits + 4;
                    end
         end
    end

    spectralEffIndoor(i) = nbits/length(SNRIndoor);
    nbits = 0;
    for j = 1:length(SNROutdoor)
          if ((SNROutdoor(j) >= 10.61) && (SNROutdoor(j) < 13.94))   %QPSK
                nbits = nbits + 2;
             else if ((SNROutdoor(j) >= 17.25) && (SNROutdoor(j) < 20.4))   %16 QAM
                        nbits = nbits + 4;
                 end
          end
    end
    spectralEffOutdoor(i) = nbits/length(SNROutdoor);
end
end
```