Rahmaan Lodhia

GTID: 902502749

ECE 4271

Project #2a

Binary Communication Transceiver Design

1. Transceiver Algorithm

1.1. Overview

The transceiver simulation is composed of two main processes, which are the transmitter stage and the receiver stage. The top-level file (transceiver.m) takes in a specified length of bits, an SNR value in dB, and a size for the modulation constellation. With these values, the program first creates a stream of bits to simulate a signal (DataGeneration.m). The simulated bits are then sent to the transmitter stage (modulation.m), which will encode the bits into symbols as defined by the specified QAM constellation. Then from the transceiver, the encoded signal will have white Gaussian noise with a standard deviation of one added to it (NoiseGeneration.m), corrupting the signal. The corrupted signal will enter the receiver stage, where it will first be mapped to estimated symbols in order to eliminate the noise involved (receiver.m). Then, the estimated symbols will be decoded into bits (demodulation.m), and it will send the estimated signal to the main program. The program will take the resulting estimated signal and compare it to the original counting the number of bit errors. Finally, it will take the total number of errors and divide by the total number of bits to arrive at the bit error rate (BER).

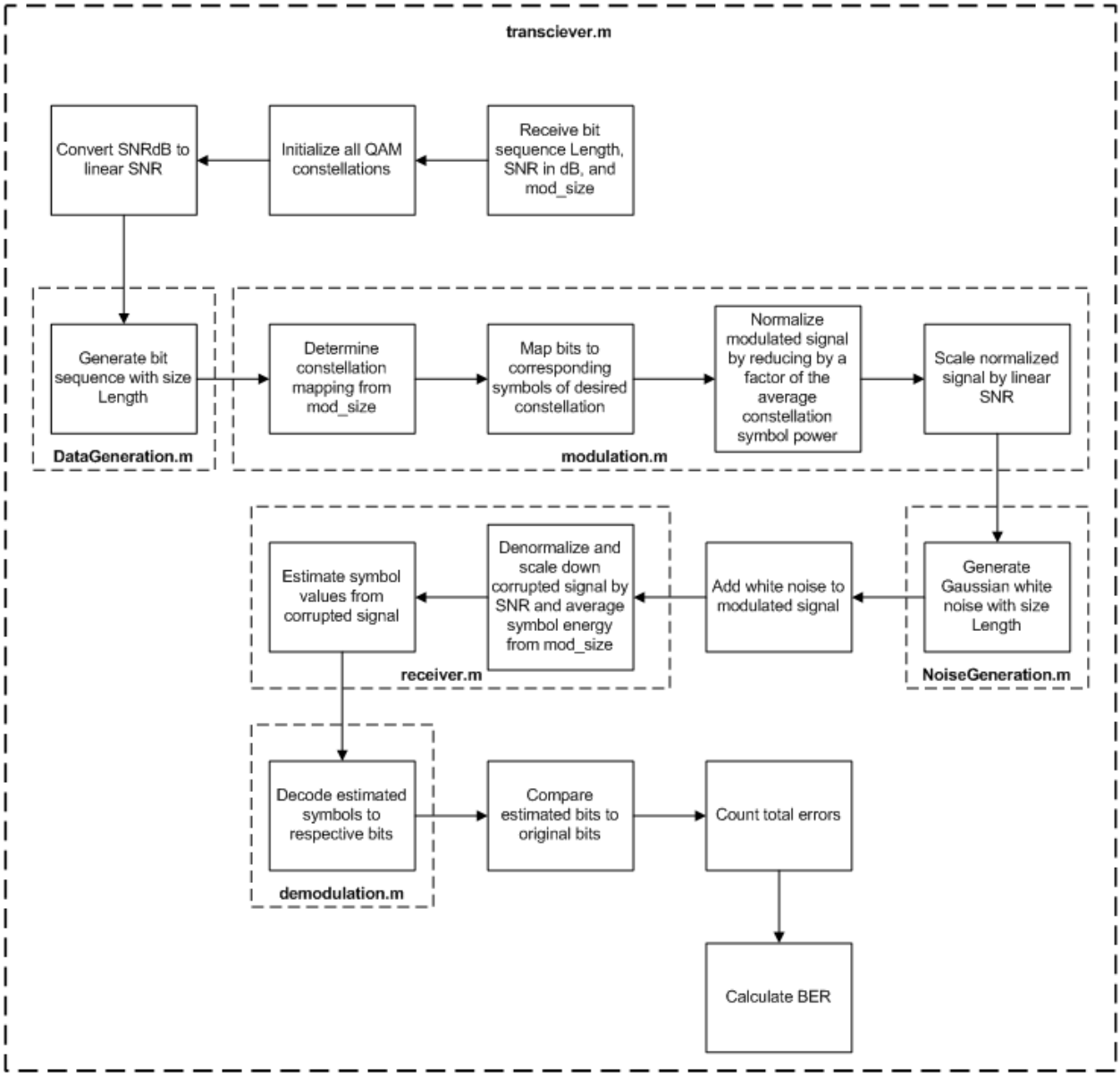
1.2. Transmitter Design

The transmitter takes in three variables: the vector of bits representing the signal, the linear SNR, and the size of the constellation to use for encoding. The transmitter stage takes place in the modulation.m file. The function will first determine from the constellation size how many bits compose a symbol. Then, it will iterate through the signal and encode each sample of bits into the proper QAM constellation mapping. When the signal has been encoded, the function will then take the encoded signal and normalize it by dividing the signal by the average signal energy of the constellation type. Taking the normalized signal, the function will then scale the signal by the symbol power as defined from the SNR, and this output signal will be sent through the channel, where noise is added, into the receiver.

1.3. Receiver Design

After the signal is corrupted by the addition of white noise from the NoiseGeneration.m function, the signal is sent to the receiver. The receiver process starts in the receiver.m function. The receiver was designed under the assumption that it would know both the SNR of the signal and the type of modulation used for the signal. These two values were passed as parameters into the receiver.m file. Within this function, the function determines which modulation symbol mapping to use to compare the corrupted signal with. The corrupted signal is then scaled down by the symbol power defined by the SNR and the average symbol power for the modulation size to a normalized, corrupted signal. When the modulation is selected, the function will then iterate through each corrupted signal and compare it against each symbol of the constellation mapping, and it will determine which of the symbols is closest to the corrupted signal. This signal will be considered the estimated value for the corrupted signal. This process will continue until a vector estimating the correct values for the corrupted signal is created, which will eliminate the noise. This estimated signal will then be sent into the demodulation.m function, where each symbol will be converted to their corresponding bits as defined by the mapping. This final sequence of estimated bits is sent back to the top-level program, where it will calculate the resulting bit-error rate.

2. Transceiver Algorithm Block Diagram



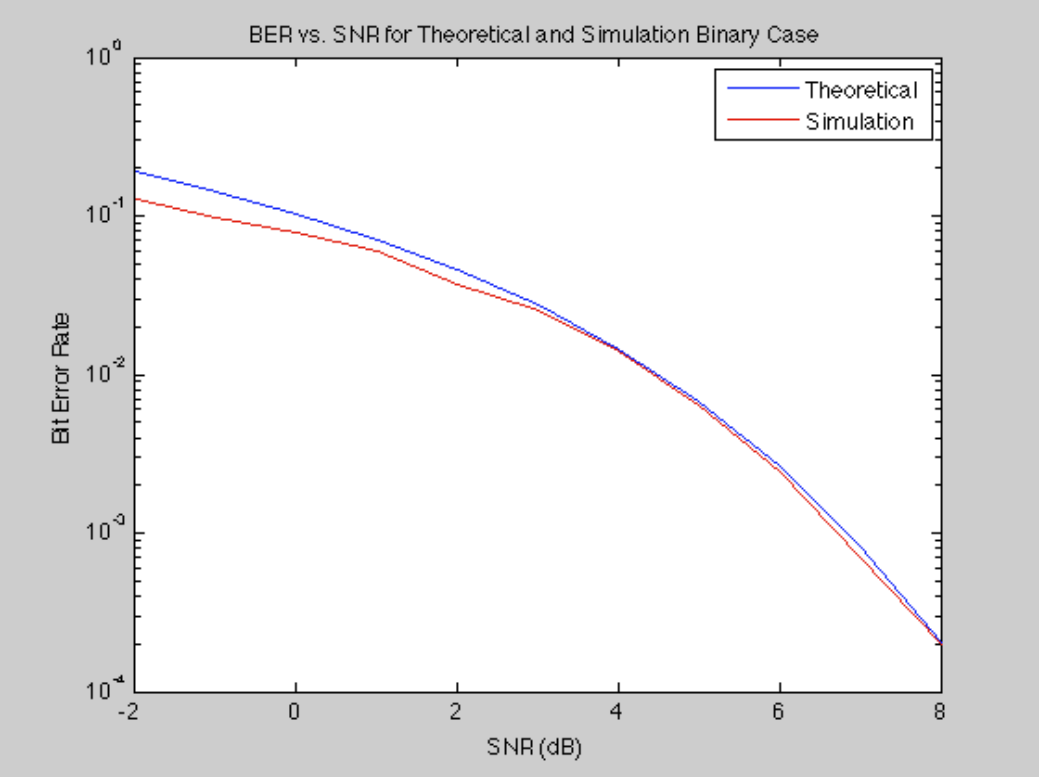
3. Transceiver Considerations

3.1. Benchmark and Resulting BER for Constellations

The transceiver simulation results were tested against the analytical results from the Q-function. The Q-function is approximately defined as:

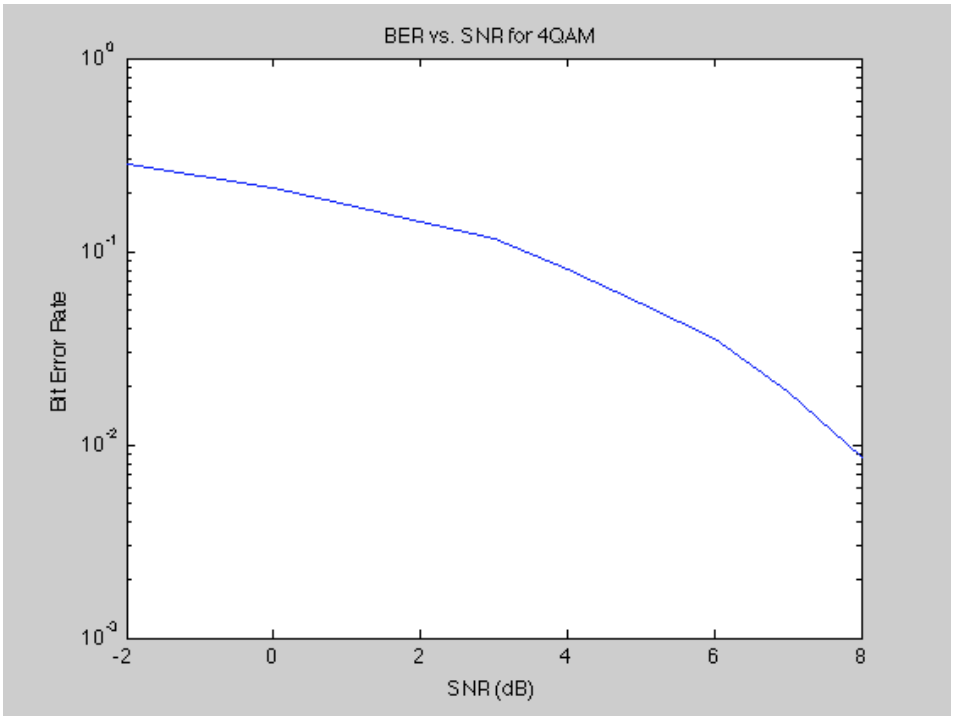
$$Q(u) \approx \frac{e^{-\frac{u^2}{2}}}{u\sqrt{2\pi}}$$

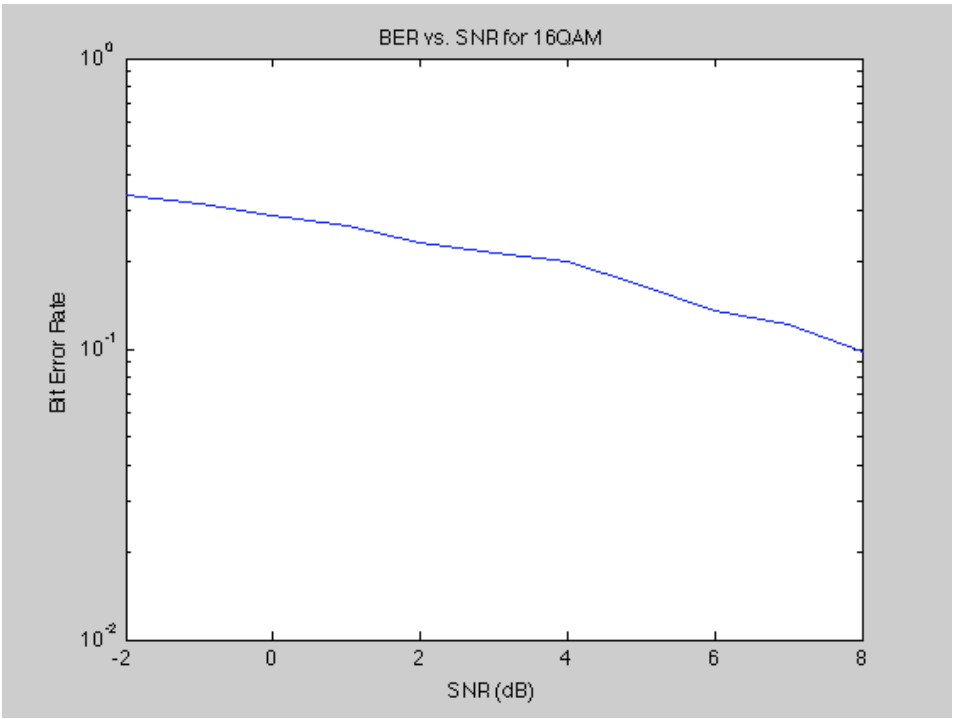
To find the theoretical BER, we used $Q\left(\sqrt{\frac{2E_b}{N_0}}\right)$ which simulated the BER vs. SNR for the binary constellation mapping case. These theoretical results were the benchmark for testing the efficiency of the transceiver algorithm. The BPSK should result in BER values at the respective SNR range of -2dB to 8dB that match closely to the benchmark case to test the accuracy of the algorithm. Below is the graph of the theoretical output with the simulation binary case.



Both curves match very close at higher SNR values, but there are some visible differences in results at lower SNR values. These differences at lower SNR values are due to the fact that Q-function formula used is an approximation of the actual Q-function, and the actual values

are lower than the approximation. Therefore, the resulting simulation for the binary case can be considered to meet the expected Q as it is slightly lower than the approximation, which means the algorithm meets the base case. The 4QAM and the 16QAM BER vs. SNR graphs are as follows:





3.2. Bit Sequence Length

The lowest possible error found for the range of SNR (in dB) was found to be in the magnitude value of 1.0e-4. This leads to the fair assumption that the bit sequence for simulation must be at least 1.0e4 for a proper, accurate measurement for the BER to be made. This sequence length was used for all BER measurements.

3.3. Signal-to-noise Ratio

The definition of the linear SNR used for the algorithm is

$$SNR = \frac{Signal\ Power}{Noise\ Power} = \frac{E_b}{N_0}$$

where E_b is the signal power per symbol band and N_0 is the power for noise. This definition is applicable for the algorithm as the program uses higher constellations. In addition since this program only simulates Gaussian white noise with a standard deviation of 1, the signal power per symbol can be considered approximately equivalent to the SNR as the noise power will be approximately equivalent to 1. This assumption is used to scale the signal in the transmitter to the specified SNR. The transmitter will scale the signal by both the linear SNR and the average energy per symbol E_{avg} of the specified QAM constellation. The resulting signal of the transmitter after noise is added is:

$$S = \frac{\sqrt{E_b}}{\sqrt{E_{avg}}} s(t) + n(t)$$

Through this method, the receiver will obtain a signal equivalent to the scaled signal plus noise. The receiver will then scale down the corrupted signal by the inverse of the factor used by the transmitter and arrive at a signal that is equivalent to:

$$S_{corrupted,normalized} = s(t) + \frac{\sqrt{E_{avg}}}{\sqrt{E_b}} n(t)$$

This resulting signal has limited the noise in the signal, and it will be tested to estimate the symbols the signal closely resembles.

3.4. Decision Logic

When the receiver is passed the corrupted signal, it will first determine which symbols it has received to use as an estimate for the data. The receiver will make this decision by

taking each corrupted symbol and comparing it against the standard symbols for the specified constellation. Each comparison will calculate the magnitude between the corrupted symbol and the standard symbols through the means of the distance formula. The lowest magnitude of error between a corrupted signal and a standard symbol will be found, and the corresponding symbol matched to it will be set as the estimated symbol for the corrupted signal. This process will repeat for each corrupted symbol until an estimated symbol vector is created.

4. Matlab Code

4.1. transceiver.m

```
function BER = transceiver(length, SNRdB, mod_size)
%Simulate Transceiver with given data length, SNR (dB), and constellation
% size.
% Input: length (length of bits)
%        SNRdB (Desired SNR for Signal)
%        mod_size (2, 4, or 16 constellation size)
%
% Output: BER (Bit Error Rate)


%Initialize Constellation Tables and Average Signal Powers
%Set as global variables for use in all functions as they are constant.
global BPSKVector QAM4Vector QAM16Vector BPSKAvgPow QAM4AvgPow ...
    QAM16AvgPow

BPSKVector = [1, -1];
QAM4Vector = [1+j, -1+j, -1-j, 1-j];
QAM16Vector = [-3+3j, -3+j, -3-3j, -3-j, -1+3j, -1+j, -1-3j, -1-j, ...
    3+3j, 3+j, 3-3j, 3-j, 1+3j, 1+j, 1-3j, 1-j];


BPSKAvgPow = sum(abs(BPSKVector).^2)/size(BPSKVector,2);
QAM4AvgPow = sum(abs(QAM4Vector).^2)/size(QAM4Vector,2);
QAM16AvgPow = sum(abs(QAM16Vector).^2)/size(QAM16Vector,2);

%Generate Random Vector of Bits
Bits = DataGeneration(length);

%Convert SNR from dB
SNR = 10^(SNRdB/10);

%Take Resulting Bits and Modulate Signal According to Given Parameters
S = modulation(Bits, SNR, mod_size);

%Generate Noise with Size Equal to Converted Signal
Noise = NoiseGeneration(size(S,2));

%Estimate Symbols from Corrupted Signals
est_sym = receiver(S+Noise, SNR, mod_size);

%Convert Estimated Symbols to Bits
est_Bits = demodulation(est_sym, mod_size);

%Calculate Difference and Total Errors
diff = Bits - est_Bits;
totalError = sum(abs(diff));

%Calculate Bit Error Rate
BER = totalError/size(Bits,2);

end
```

4.2. DataGeneration.m

```
function Bits = DataGeneration(length)
%Generate random vector of 1s and 0s
```

```
Bits = randi(2, 1, length) - 1;
end
```

4.3. modulation.m

```
function S = modulation(Bits, SNR, mod_size)
%Take in vector of Bits and modulate signal to respective signals according
%to mod size.
%Inputs: Bits (Vectors of 1s and 0s)
%        SNR (Signal Noise Ratio - Linear Value)
%        mod_size (2, 4, or 16 constellation size)
%
%Outputs: S (Converted and modulated Signal)

%Gain access to global variables
global BPSKVector QAM4Vector QAM16Vector BPSKAvgPow QAM4AvgPow ...
    QAM16AvgPow

%Initialize Variables
convertedBits = [];
unNormS = [];
normS = [];
step = log(mod_size)/log(2);
symbolVector = [];
symbolPower = [];

%Determine which Constellation Values to use
switch mod_size
    case 2
       symbolVector = BPSKVector;
       symbolPower = BPSKAvgPow;
    case 4
       symbolVector = QAM4Vector;
       symbolPower = QAM4AvgPow;
    case 16
       symbolVector = QAM16Vector;
       symbolPower = QAM16AvgPow;
end

%Iterate through Bits according to mod_size
for i = 1:step:length(Bits)
    %Check if enough bits remain to convert to a symbol
    if length(Bits(i:end)) >= step
        sample = Bits(i:i+step-1);
        %Convert sample of bits to respective symbol of constellation
        convertedBits = [convertedBits symbolVector(bin2dec(num2str((sample)))+1)];
    end
end

%Normalize converted values by the average symbol power for selected
%constellation
normS = convertedBits ./ sqrt(symbolPower);

%Scale normalized signal by signal power as determined from SNR
S = sqrt(SNR) .* normS;

end
```

4.4. NoiseGeneration.m

```
function Noise = NoiseGeneration(length)
%Generate White Gaussian Noise with standard deviation of 1 with size of
%length.
Noise = (randn(1, length) + j.*randn(1, length)) ./ sqrt(2);
End
```

4.5. receiver.m

```
function est_sym = receiver(signal, SNR, mod_size)
%Take in corrupted signal and estimate the closest symbols for the
%corrupted data
%Inputs: signal (Corrupted signal S+Noise)
%        SNR (Signal Noise Ration - Linear Value)
%        mod_size (2, 4, or 16 constellation size)
%
%Outputs: est_sym (Estimated symbols)


%Gain access to global variables for constellations
global BPSKVector QAM4Vector QAM16Vector BPSKAvgPow QAM4AvgPow ...
    QAM16AvgPow

%Initialize variables
est_sym = [];
errMag = [];
testSymbol = [];
testPower = [];

%Determine which constellation data to use
switch mod_size
    case 2
        testSymbol = BPSKVector;
        testPower = BPSKAvgPow;
    case 4
        testSymbol = QAM4Vector;
        testPower = QAM4AvgPow;
    case 16
        testSymbol = QAM16Vector;
        testPower = QAM16AvgPow;
end

%Normalize corrupted signal with average signal power and SNR
normSignal = signal ./ (sqrt(SNR / testPower));

%Iterate through corrupted signal
for symbol = normSignal
    %Measure error magnitude between signal and symbols for constellations
    errMag = sqrt(((real(symbol) - real(testSymbol)).^2 + (imag(symbol) -
imag(testSymbol)).^2));

    %Find the symbol with the minimum error and add that as the estimated
    %symbol.
    [minErr index] = min(errMag);
    est_sym = [est_sym testSymbol(index)];
end

end
```

4.6. demodulation.m

```
function est_bits = demodulation(est_sym, mod_size)
%Take in estimated symbols and convert them back to bits.
%Inputs: est_sym (Vectors of estimated symbols)
%        SNR (Signal Noise Ration - Linear Value)
%        mod_size (2, 4, or 16 constellation size)
%
%Outputs: est_bits (Estimated bit values)


%Gain access to global variables for constellations
global BPSKVector QAM4Vector QAM16Vector BPSKAvgPow QAM4AvgPow ...
    QAM16AvgPow

%Initialize variables
bitLength = log(mod_size)/log(2);
symbolVector = [];
est_bits = [];

%Determine which constellation to use
switch mod_size
    case 2
        symbolVector = BPSKVector;
    case 4
        symbolVector = QAM4Vector;
    case 16
        symbolVector = QAM16Vector;
end

%Iterate through estimated symbols
for sym = est_sym
   %Convert symbol to index integer and then to binary string
   index = find(sym == symbolVector);
   binNumber = dec2bin(index-1, bitLength);
   %Convert string to number and add to est_bits
   for i = 1:length(binNumber)
       est_bits = [est_bits str2num(binNumber(i))];
   end
end

end
```