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 modulation 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**

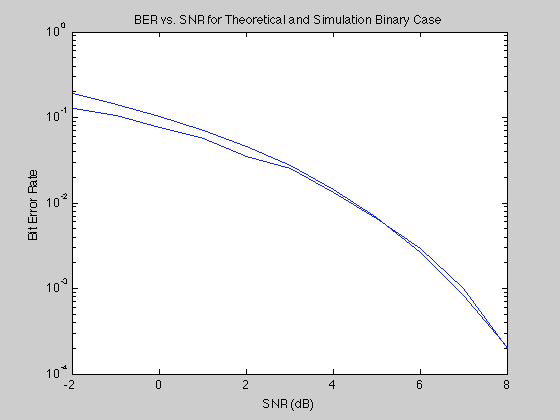
The transmitter takes in three variables: the vector of bits representing the signal, the linear SNR, and the size of the constellation to use for encoding. The transmitter stage takes place in the modulation.m file. The function will first determine from the constellation size how many bits compose a symbol. Then, it will iterate through the signal and encode each sample of bits into the proper 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 calcuate the resulting bit-error rate.**2. Transceiver Algorithm Block Diagram**

**3. Transceiver Considerations**

* 1. **Benchmark and Resulting BER for Constellations**



* 1. **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.

* 1. **Signal-to-noise Ratio**

The definition of SNR used for the algorithm is stated below:

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. This assumption is used to scale the signal, as it is being transmitted and received as a corrupted signal. Through this method, the receiver will obtain a signal equivalent to:. This resulting signal will be tested to estimate the symbol it closely resembles.

* 1. **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. 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**

* 1. **transceiver.m**

function digits = tt\_decode(x)

%tt\_decode.m

%Take in a signal composed of dial tones, decode signal, and return a

%number in the format of NNN-NNN-NNNN.

%Set Sampling Frequency

Fs = 8000;

%Set Analysis Window Size and Time Division

windowSize = 200;

dt = windowSize/Fs;

%Initialize duration variables

toneDuration = 0;

silenceDuration = 0;

%Initialize Digit and Tone variables

currentDigit = 'None';

previousDigit = 'None';

previousTone = '';

%Initialize vector to hold found digits

foundDigit = [];

%Iterate Over Signal using set window size

for startIndex = 1:windowSize:length(x)-1

%Check if enough samples are preset to allow for window analysis

if (length(x) - startIndex >= windowSize)

%Create windowed sample from signal

sample = x(startIndex:startIndex+windowSize-1);

%Calculate dual frequencies found in sample if any

[freq1 freq2] = calcFreq(sample);

%Check result. If 0, there is no tone. Otherwise, decode the

%frequencies and find digit.

if (freq1 == 0 && freq2 == 0)

currentDigit = 'None';

else

currentDigit = lookupDigit(freq1, freq2);

end

%Check if the current tone found in the window matches the previous

%tone found in the last window.

if(strcmp(previousDigit, currentDigit))

%Tones are the same and consistent

if(strcmp(currentDigit, 'None'))

%Samples are silence in a row. Increase silence duration

silenceDuration = silenceDuration + 2\*dt;

else

%Samples are same tone. Increase current tone duration

toneDuration = toneDuration + dt;

end

elseif(~strcmp(previousDigit, 'None') && (strcmp(currentDigit,'None')))

%Tones are different. Possible that tone has finished or noise

%has created an illusion of silence, so store the previous tone

%in mememory, and start measuring silence duration.

silenceDuration = 0;

previousTone = previousDigit;

elseif(strcmp(previousDigit, 'None'))

%Silence is over. Tone is detected. Reset silence duration.

silenceDuration = 0;

if (~strcmp(previousTone, currentDigit) && ~strcmp(previousTone, ''))

%If current tone is different than stored tone and the

%stored tone exists, this current tone is a wrong

%measuremnt due to noise. Ignore this sample.

currentDigit = previousTone;

toneDuration = toneDuration + dt;

end

end

%Check Silence Duration

if(silenceDuration >= .05)

%Silence duration is long enough to be a pause between tones.

%Previous tone is potentially a valid digit

if(toneDuration >= 0.04 && toneDuration <= .40001)

%Check if tone duration of previous tone is valid. If so,

%then a valid dial tone is added to the vector of tones.

%Reset duration and tone variables.

foundDigit = [foundDigit previousTone];

previousTone = '';

toneDuration = 0;

silenceDuration = 0;

end

end

end

%Store current digit tone as previous digit tone

previousDigit = currentDigit;

end

%Take resulting found digits and format output.

%Will return error if not enough digits are found.

digits = formatDigit(foundDigit);

end

* 1. **DataGeneration.m**

function [freq1, freq2] = calcFreq(signal)

%calcFreq.m

%Calculate Frequencies in the given signal, and determine which frequencies

%for dial tones are valid and present if any.

%Initialze N and k

N = 205;

k = [18, 20, 22, 24, 31, 34, 38, 42];

goertzelOutput = [];

%Iterate through different k values, and determine Goertzel DFT outputs at

%each k.

for j = k

goertzelOutput = [goertzelOutput gfft(signal, N, j)];

end

%Calculate peaks from low and high bands

[peak1 ind1] = max(goertzelOutput(1:4));

[peak2 ind2] = max(goertzelOutput(5:8));

%Calculate low and high indices

low = k(ind1);

high = k(ind2+4);

%Calculate the validity of the Fundamental Frequency levels and the Harmonic

%Frequency levels for the given signal at the maximum frequencies.

fundValid = checkFundamentalLevel(peak1, peak2, goertzelOutput);

harmValid = checkHarmonicLevel(signal, goertzelOutput, ind1, ind2+4);

%Check valid variables

if (harmValid && fundValid)

%If signal is valid in fundamental frequency strength and has

%negligible harmonics, it is a valid signal, and its low band and high

%band frequencies are calculated.

switch low

case 18

freq1 = 697;

case 20

freq1 = 770;

case 22

freq1 = 852;

case 24

freq1 = 941;

end

switch high

case 31

freq2 = 1209;

case 34

freq2 = 1336;

case 38

freq2 = 1477;

case 42

freq2 = 1633;

end

else

%Signal is not valid. So, return 0.

freq1 = 0;

freq2 = 0;

end

end

* 1. **modulation.m**

function valid = checkFundamentalLevel(peak1, peak2, fundamentalDFT)

%checkFundamentalLevel.m

%Checks the power lever of the fundamental frequencies in the signal.

%Returns 1, if this signal is a valid tone. Returns 0, otherwise.

%K is a threshold. The magnitude square of the peaks must be K times

%greater than the sum of the magnitude squares of the remaining frequencies

%in the band for this signal to have enough power to be considered a tone

%and not noise.

k = 7;

valid = ( ( abs(peak1)^2 >= k\*(sum((abs(fundamentalDFT(1:4)).^2)) - abs(peak1)^2) ) ...

&& ( ( abs(peak2)^2 >= k\*(sum((abs(fundamentalDFT(5:8)).^2)) - abs(peak2)^2) ) ) );

end

* 1. **NoiseGeneration.m**

function valid = checkHarmonicLevel(signal, fundamentalDFT, ind1, ind2)

%checkHarmonicLevel.m

%Checks the Harmonics present in the signal. Returns 1, if harmonics are

%negligible. Returns 0, if harmonics are too high indicating that this is

%not a pure tone.

%Initializes N and k

N = 201;

k = [35, 39, 43, 47, 61, 67, 74, 82];

harmonics = [];

%Sets a threshold value. If the sum of the magnitude square of harmonics

%is less than this value, then the harmonics are negligible.

thresh = 150;

%Iterate through k values, and calculate DFT for each harmonic frequency

for j = k

harmonics = [harmonics gfft(signal, N, j)];

end

%Calculate the ratio of harmonic to fundamental for the max values. If the

%ratio is too high, then this signal is not a pure tone.

ratio = mean(abs(harmonics([ind1 ind2])) ./ abs(fundamentalDFT([ind1 ind2])));

%Valid if the harmonics meet one of these condiditons.

valid = (sum(harmonics.^2) < thresh) || (ratio < .15);

end

* 1. **receiver.m**

function digit = lookupDigit(f1, f2)

%lookupDigit.m

%Takes frequencies and returns the corresponding digit for those two

%frequencies.

%Initialize Digit Lookup Table

digitTable = ['1', '2', '3', 'A';

'4', '5', '6', 'B';

'7', '8', '9', 'C';

'\*', '0', '#', 'D'];

%Using given frequencies, determine row and column of digit table

switch f1

case 697

i = 1;

case 770

i = 2;

case 852

i = 3;

case 941

i = 4;

end

switch f2

case 1209

j = 1;

case 1336

j = 2;

case 1477

j = 3;

case 1633

j = 4;

end

%Find the indicated digit

digit = digitTable(i,j);

end

* 1. **demodulation.m**

function digits = formatDigit(digitVector)

%formatDigit.m

%Takes a string of digits and formats the input into a phone number

%If the string of digits is too short, then return 'Error' indicating that

%a valid phone number is not returned.

if length(digitVector) < 10

digits = 'Error';

else

digits = [digitVector(1:3) '-' digitVector(4:6) '-' digitVector(7:10)];

end