

School of Computer Science and Engineering (SCSE)

CE3006: Digital Communications

*Project Report*

*Agnesh Ramesh* (U1921141K)

*Kesarimangalam Srinivasan Abhinaya* (U1923302J)

*Singh Aishwarya* (U1923952C)

*Unnikrishnan Malavika* (U1923322E)

*Vedula Kartikeya* (U1923891G)

**Table of Contents**

1. Project Scope and Objective
2. Introduction
3. Digital Communication System Overview
4. Implementation
   1. Running the code
   2. Phase 1: Data Generation
   3. Phase 2: Modulation for Communication
      1. On – Off Keying (OOK)
      2. Binary Phase Shift Keying (BPSK)
      3. Binary Frequency Shift Keying (BFSK)
   4. Phase 3: Error Control Coding
5. Results
   1. Phase 1: Data Generation
      1. Observations
      2. Evaluation
   2. Phase 2: Modulation for Communication
      1. Observations
      2. Evaluation
   3. Phase 3: Error Control for Coding
      1. Observations
      2. Evaluation
6. Conclusion
7. Scope for Improvement
8. Bibliography
9. **Project Scope and Objective**

The goal of this course project is to create a simple digital communication system employing MATLAB and investigate its functioning through various digital communication processes.

1. **Introduction**

The conveyance of information from one place to another via a multitude of methods is referred to as communication. It entails the generation of a message signal, the precise description of that signal, the encoding of symbols in a form suitable for transmission over a physical medium of interest, the transmission of encoded symbols to the desired destination, the decoding and reproduction of original symbols, and the recreation of original message signal with a definable degradation in quality.

Communication systems are divided into two types: digital and analog. Analog Communication Systems use a transmission medium to send analog waveforms. They are conceptually simpler, but more complex to construct, and there is no major attempt to adapt the waveform to suit the channel distortions. However, with Digital Communication Systems, the transmitted signal is transformed to digital, which is complicated conceptually but simple to implement. It is feasible to identify a finite set of waveforms that fit channel properties and are thus more resistant to channel distortions. It is capable of efficient and reliable transmission but necessitates a significant quantity of electrical hardware.

Digital communication techniques are utilized. Digital signals are less susceptible to noise and distortion. They can be perfectly rebuilt over great distances by regenerative repeaters, avoiding error propagation. Digital data from several sources may be readily multiplexed. They can also be encoded to rectify mistakes and encrypted for privacy. Digital circuits that are reasonably cheap can be employed.

This project report will explore the implementation of a simple digital communication system – from generating the data to modulating it for transmission. It also explores methods to improve performance by using error control coding.

1. **Digital Communication System Overview**

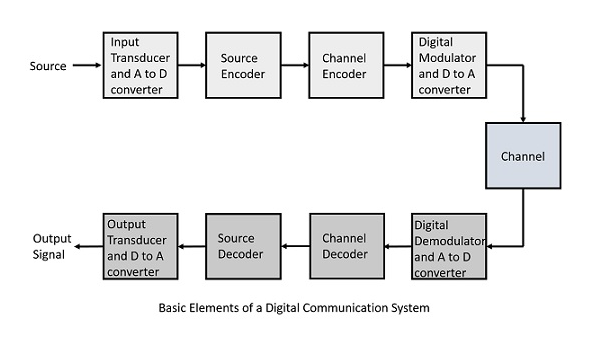
The three main components of a digital communication system are as follows –

* Transmitter
* Channel
* Receiver

The signal is received by the transmitter from the source. The sent signal is then routed through the channel, where it travels to the receiver before being delivered to the user. A signal is a mathematical function-based representation of a physical quantity represented as a function of time. In addition, a system is any physical equipment that generates an output signal in response to an input signal. Digital communication systems are thought to be linear time invariant. This means it has a linear response in both time and frequency. In a digital communication system, the channel is considered to be additive in nature, with other channel effects being disregarded.

On the transmitter side, the signal from the source is routed through an analog to digital converter, which samples, quantizes, and converts the raw analog signals to digital bits for transmission. After that, the digital signal is encoded and modulated (bandpass level) to broadcast at a higher carrier frequency. Continuing, the same signal is demodulated on the receiver side to get the original frequency, decoded, and sent through a digital to analog converter to retrieve the original signal that was broadcast by the source.

Diagram

Description automatically generatedThis entire process can be represented in Figure 1 below. The scope of this project is to simulate the steps depicted in the figures and explore the different techniques used throughout the digital communication process to transmit signals from source to the receiver.

**Figure 1: Elements and Process of a Digital Communication System**

1. **Implementation** 
   1. **Running the code**

To demonstrate the functionality of the digital communication system, run the following files in the given order -

* 1. **Phase 1: Data Generation**

This phase contains information about baseband modulation and demodulation. The binary bitstream will be produced here and broadcast over an additive white Gaussian noise channel with varying SNR values. The performance of the bit error rate will be analysed and displayed against various SNR levels.

First, we declare variables that will be used in data generation. It is given that the number of bits for transmission is 1024 (which indicates that N = 1024). Then, a series of random binary digits (0 or 1) is generated. Furthermore, binary digits are converted to 1 (that is, 1 to +1 and 0 to -1). This is the information the system will send. Next, an equal amount of noise samples is produced. The produced noise has a normal distribution with a mean of zero and a variance of one unit. The noise variance in relation to the SNR (signal to noise ratio) value is then modified. Finally, the SNR value is set to that effect. These definitions are declared in the commented MATLAB code below –

**%Number of bits = 1024 (given)**

nBits = 1024;

**%Generating random binary digits**

Data = randi([0 1],1,nBits);

**%Converting to +1 and -1**

Signal = 2 .\* (Data - 0.5);

**%Generating noise with mean = 0 and variance = 1**

Noise = randn(1,nBits);

**%Fixing SNR value to 10 dB**

SNR = 0:5:50;

**%Creating a result array to store error for each value of SNR**

Result = zeros([1 11]);

**%Current index to track result**

index = 1;

Following that, SNR is utilized to create noise variance, which is expressed in decibels. The signal (the input data) is assumed to have unit power (S=1). The noise variance is then calculated and used in conjunction with the noise samples to produce the desired noise. The noise samples are then mixed in with the data being sent. This is the signal that was received.

**%Looping over the different values of SNR**

for i = 1:length(SNR)

currSNR = SNR(i);

**%SNR = 10log(S/N), where S = Signal Power, N = Noise Power**

**%Signal has unit power -> S = 1**

**%Therefore, using above equation, N = 0.1**

noisePower = 0.1.^(currSNR/10);

**%Adjusting Noise based on the SNR value**

Noise = sqrt(noisePower) .\* Noise;

**%Calculating the received signal**

Received = Signal + Noise;

At the receiver, a threshold logic is then evaluated. This value is set to 0. If the received signal is greater than or equal to the threshold, it is assigned a value of 1, otherwise it is assigned a value of 0. The bit error rate is then calculated. The threshold logic's output values are compared to the binary digits input. The processes from noise addition through bit error rate computation are repeated for different SNRs.

**%Fixing a threshold value for the received signal**

Threshold = 0;

**%Calculating the output based on the threshold level**

Output = zeros(1,nBits);

for k = 1 : nBits

if Received(k) > Threshold

Output(k) = 1;

else

Output(k) = 0;

end

end

**%Computing the total number of errors**

Error = 0;

for k = 1 : nBits

if Output(k) ~= Data(k)

Error = Error + 1;

end

end

**%Calculating the bit error rate**

**%BER = (Total errors)/(Number of bits)**

BER = double(Error)/double(nBits);

Result(index) = BER;

index = index + 1;

* 1. **Phase 2: Modulation for Communication**

This section contains information about band-pass modulation and demodulation. The team has implemented several band-pass modulation algorithms in the project. Simpler modulation methods like On-Off Keying and Binary Phase Shift Keying have been developed as base modulation techniques. In addition to the basic algorithms, the team has also implemented Binary Frequency Shift Keying to explore the impact of different modulation techniques in communication systems.

* + 1. **On-Off Keying (OOK)**

The simplest kind of amplitude-shift keying (ASK) modulation is on–off keying (OOK), which expresses digital data as the presence or absence of a carrier wave. In its most basic form, the presence of a carrier for a certain period indicates a binary one, whereas the lack of a carrier for the same duration represents a binary zero.

The first step is to declare all variables that are to be used in the developed algorithm. From the previous phase, the number of bits for transmission remains the same at 1024 bits. The carrier frequency is given as 10kHz, where the carrier signal is oversampled by 16 times, as defined by the variable carrierSignal. The baseband data rate is given as 1kbs. Next, the sampling period formula is defined in the code as the ratio of the carrier signal and the data rate. Then, a low pass butterworth filter is implemented using a MATLAB library function butter. The amplitude is then set as 1, and the time period range is set, as defined in the equation given in the MATLAB code (t). Additionally, the Signal to Noise Ratio is declared (in dB as well), and the error rate of the OOK algorithm is initialised to an array of zeros with the length of SNR. Finally, the carrier is defined, for demodulation by the formula given as amp.\*cos(2\*pi\*carrierFrequency\*t)and the length of the signal, along with the number of runs of this algorithm are declared in the code snippet shown below -

nBits = 1024;

carrierFrequency = 10000;

carrierSignal = carrierFrequency \* 16;

dataRate = 1000;

samplingPeriod = carrierSignal / dataRate;

[lowB, lowA] = butter(6,0.2);

amp = 1;

t = 0: 1/carrierSignal : nBits/dataRate;

SNR\_dB = 0:5:50;

SNR = (10.^(SNR\_dB/10));

modifySNR\_dB = 5;

errorRateOOK = zeros(length(SNR));

carrier = amp .\* cos(2\*pi\*carrierFrequency\*t);

signalLength = carrierSignal\*nBits/dataRate + 1;

numRuns = 10;

The next step was to design a function for simulating a decision device and sampling. The sampling function sample takes in 3 parameters – the output signal of the low pass filter, the sampling period and the number of bits. The input signal is then sampled to produce a sampled signal which is then passed to the decision device, where the created function is called. The decision\_device function takes in 3 parameters as well, namely the sampled signal, the number of bits and the threshold value (half of amplitude in this case). The output of the decision device is then used for error calculations. The code snippet below depicts the two functions -

**% Sampling and Decision Device Simulation**

function sampled = sample(x,sampling\_period,num\_bit)

sampled = zeros(1, num\_bit);

for n = 1: num\_bit

sampled(n) = x((2 \* n - 1) \* sampling\_period / 2);

end

end

function binary\_out = decision\_device(sampled,num\_bit,threshold)

binary\_out = zeros(1,num\_bit);

for n = 1:num\_bit

if(sampled(n) > threshold)

binary\_out(n) = 1;

else

binary\_out(n) = 0;

end

end

end

The final step is to put everything together and create the OOK algorithm. As depicted in Figure 1 of Digital Communication System Overview, there are different steps that are involved in modulation and demodulation of a signal. In the OOK algorithm, first, the noise is generated, then, the transmitted signal is obtained (sum of signal and noise). Next, the transmitted signal is passed to a non-coherent detector – square law detector – and is then passed through a low pass filter, sampled, and the final signal is retrieved. Lastly, the error is calculated. The code snippet below depicts the working of the aforementioned OOK algorithm –

**% Noise Generation**

OOKSignalPower = (norm(OOKSignal)^2)/signalLength;

OOKNoise = OOKSignalPower ./SNR(i);

OOKNoise = sqrt(OOKNoise/2) .\*randn(1,signalLength);

**% Transmit signal**

OOKTransmit = OOKSignal + OOKNoise;

**% Use non-coherent detection - square law detector**

sqLawOOK = OOKTransmit .\* OOKTransmit;

**% Pass this through the LP filter**

LPfilterOOK = filtfilt(lowB, lowA, sqLawOOK);

**% Sample the signal**

OOKsample = sample(LPfilterOOK, samplingPeriod, nBits);

finalResultOOK = decision\_device(OOKsample,nBits, amp/2);

**% Calculate Error**

errorOOK = 0;

for k = 1: nBits - 1

if(finalResultOOK(k) ~= data(k))

errorOOK = errorOOK + 1;

end

end

avgErrorOOK = errorOOK + avgErrorOOK;

* + 1. **Binary Phase Shift Keying (BPSK)**

Binary phase shift keying (BPSK) is the most primitive form of PSK, with "binary" referring to the usage of two-phase offsets (one for logic high, one for logic low). We may intuitively see that more separation between these two stages will make the system more resilient. Because we have only 360° of phase to deal with, the greatest difference between the logic-high and logic-low phases is 180°. Inverting a sinusoid is the same as shifting it by 180°; hence, BPSK is just inverting the carrier in response to one logic state and leaving it alone in response to the other logic state.

The implementation of BPSK is similar to that of OOK in section 4.3.1. For BPSK, however, the data signal needs to be represented in +1 or -1 format. Other than the given format, the steps for modulation and demodulation are the same from the OOK section, with the sampling and decision device functions being recycled for BPSK as well. The code snippet below depicts the BPSK algorithm –

bpskSourceSignal = 2.\* dataStream - 1;

bpskSignal = carrier .\* bpskSourceSignal;

bpskSignalPower = (norm(bpskSignal)^2)/signalLength;

bpskNoisePower = bpskSignalPower ./ SNR(i);

bpskNoise = sqrt(bpskNoisePower/2) .\* randn(1,signalLength);

bpskTransmit = bpskSignal + bpskNoise;

bpskDemodulated = bpskTransmit .\* (2 .\* carrier);

bpskFiltered = filtfilt(lowB, lowA, bpskDemodulated);

bpskSample = sample(bpskFiltered, samplingPeriod, nBits);

finalResultBPSK = decision\_device(bpskSample,nBits, 0);

errorBPSK = 0;

for k = 1: nBits - 1

if(finalResultBPSK(k) ~= data(k))

errorBPSK = errorBPSK + 1;

end

end

errorRateBPSK(i) = errorBPSK/nBits;

* + 1. **Binary Frequency Shift Keying (BFSK)**

The bit stream in frequency-shift keying (FSK) is represented by changes between two frequencies. In a binary frequency-shift key mechanism, the two binary states, logic 0 (low) and 1 (high), are each represented by an analog waveform. Logic 0 is represented by a certain frequency wave, while logic 1 is represented by a different frequency wave.

Following the algorithms from sections 4.3.1 and 4.3.2, BFSK algorithm has been designed to have 2 different carrier signals**,** one with a frequency of 10 kHz and the other with 5 kHz. After this step, the modulated signal is calculated by calculated based on the signal required to be transmitted. Iterating through the range of SNR values from -50 to 50 with a step size of 5, we calculate the BER for each value of SNR. For transmission, a noise with noise power based on the current iteration of SNR is added to the signal. Coherent demodulation is used on the receiver’s end. After demodulation, the signal is sampled to generate the result signal. This result signal is used to compute the bit error rate. The code snippet below demonstrates the working of BFSK –

**% Define the 2 carrier functions**

carrier1 = cos(2\*pi\*carrierFrequency\*t);

carrier0 = cos(pi\*carrierFrequency\*t);

**% Find the modulated signal**

FSKmodulated1 = carrier1 .\* (transmittedSignal == 1);

FSKmodulated2 = carrier0 .\* (transmittedSignal == -1);

modulatedSig = FSKmodulated1 + FSKmodulated2;

sigPower = rms(modulatedSig)^2;

index = 0;

SNRvalues = -50:5:50;

meanBitError = zeros([1 length(SNRvalues)]);

theoreticalError = zeros([1 length(SNRvalues)]);

for SNR = SNRvalues

index = index+1;

bitErrorRate = zeros([1 nBits]);

**% Calculate noise power from SNR**

noisePower = sigPower/(10^(SNR/10));

**% Run the experiment for each sample 20 times**

for sample = 1:20

**% Generate the noise**

Noise = randn(1,length(modulatedSig));

Noise = sqrt(noisePower) .\* Noise;

**% Find the transmitted signal after noise**

transmittedFSK = modulatedSig+Noise;

**%Coherent demodulation of FSK**

BFSK\_demod1 = transmittedFSK .\* 2.\* carrier1;

BFSK1\_filter = filtfilt(b,a,BFSK\_demod1);

BFSK\_demod0 = transmittedFSK .\*2 .\* carrier0;

BFSK0\_filter = filtfilt(b,a,BFSK\_demod0);

BFSK\_demod = BFSK\_demod1 -BFSK\_demod0 ;

count = 0;

result = zeros([1 nBits]);

­­­­­­­­

* 1. **Phase 3: Error Control Coding**

The

1. Results
2. Conclusion
3. Scope for Improvement
4. Contribution
5. Bibliography

Academic.microsoft.com. 2021. Microsoft Academic. [online] Available at: <https://academic.microsoft.com/topic/170030856/publication/search?q=On-off%20keying&qe=And(Composite(F.FId%253D170030856)%252CTy%253D%270%27)&f=&orderBy=0> [Accessed 28 October 2021].

Keying, B., 2021. BPSK - Binary Phase Shift Keying. [online] Mpirical. Available at: <https://www.mpirical.com/glossary/bpsk-binary-phase-shift-keying> [Accessed 28 October 2021].

Northern Illinois University. 2021. Binary Frequency Shift Keying - NIU - Internet Accessible Remote Laboratories. [online] Available at: <https://www.niu.edu/remote-lab/resources/graphical-user-interfaces/frequency-shift.shtml> [Accessed 28 October 2021].

Encyclopedia Britannica. 2021. frequency-shift keying | communications. [online] Available at: <https://www.britannica.com/technology/frequency-shift-keying> [Accessed 28 October 2021].

A. Froehlich, “What is frequency-shift keying (FSK)?,” SearchNetworking, 01-Oct-2021. [Online]. Available: https://www.techtarget.com/searchnetworking/definition/frequency-shift-keying. [Accessed: 28-Oct-2021].