EE 4330 Final Project Report

Chi Shing Poon

Electrical Engineering Department, University of Texas at Arlington

710 South Nedderman, Drive Arlington, Texas, United States of America

Chishing.poon@mavs.uta.edu

***Abstract*- This project demonstrates a few key principles of a digital communication system. The project is a mock digital communication system that has PCM, DPCM, Line Encoding, and BFSK implemented and applied to an image.**

I. INTRODUCTION

Technology has been rapidly growing for the past few decades. This includes the rapid expansion of digital communication techniques and applications. In these days of data and information, the process of transmitting and receiving data without error is incredibly important. Methods and theories that can turn the analog signal we were all so used to into the digital signals that are used today.

In order to effectively transmit and receive these digital signals, multiple methodologies are introduced. Some of those techniques will be demonstrated in this project. They include Quantization, Pulse Code Modulation (PCM), Differential Pulse Code Modulation (DPCM), Line Encoding, Pulse Shaping, and various binary carrier modulations.

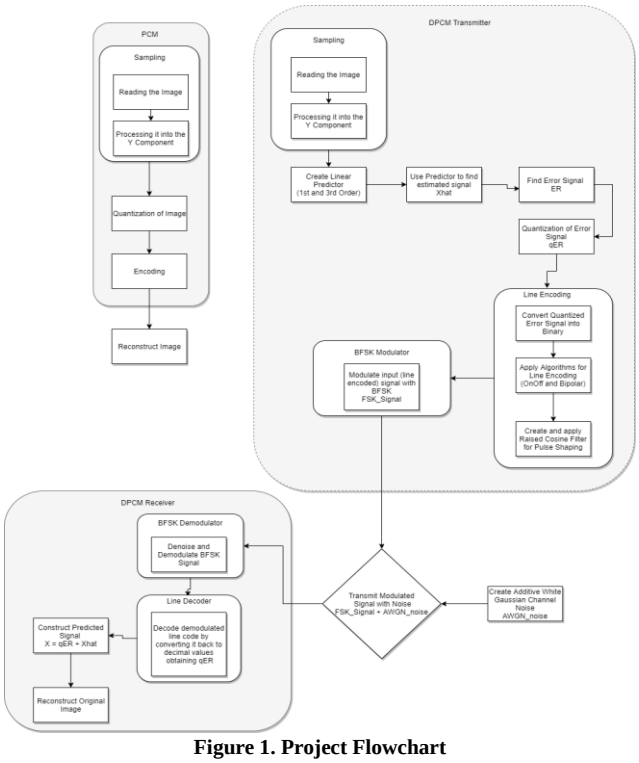

Figure 1. Project Flowchart

PCM and DPCM will be demonstrated. The basic process of PCM would be to sample, quantize, and encode the image. DPCM follow the same scheme as PCM but it will use a linear predictor in order to obtain a better result than PCM. Line encoding and BFSK modulating will also be implemented with DPCM to demonstrate a digital communication system.

II. THEORIES AN CONCEPTS

A. PULSE CODE MODULATION

The most widely used and the most important form of pulse modulation nowadays is pulse code modulation, PCM. It permits the simultaneous transmission of multiple signals on the same channel by interweaving them, given that they are modulated.

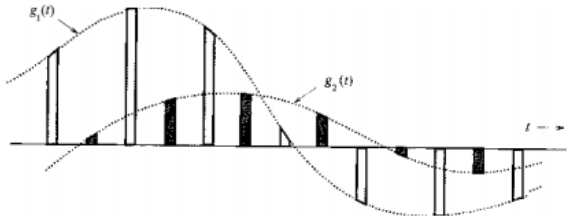

Figure 2a. Multiplexing of two signals

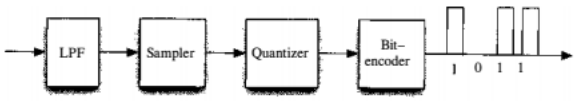

Figure 2b. Diagram for PCM System

PCM is applied when converting an analog signal into a digital signal, it converts the L-ary signal into a binary signal by sending pulses from the transmitter to the receiver. The three steps of PCM is: sampling, quantization, and encoding, as shown in figure 2.

Sampling a signal is the process collecting analog data and storing those data in its digital form. Sampling theorem explains that a signal should be sampled at, at least, twice the

highest frequency in its signal spectrum in order for its signal to be reconstructed successfully.

Quantization allow each sample to be approximated to the nearest quantized level without compromising the signal quality. There are different levels of quantization when this practice is used, and the number of levels are the quantized partitions of the signal range. The higher the level, the more information can be retained or captured from the signal. For example, when the values of a signal are originally represented by 8 bits, it is able to represent its true values. When that signal goes through quantization with a level of 16 (4 bits), its information gets compressed and represented in 4 bits instead of 8 bits. Consider an analog signal m(t) that lies in the range $(-mp, mp)$ and is partitioned into L sublevels, each having a magnitude of $\Delta v = 2mp/L$. The amplitude of each sample should be the midpoint value of their corresponding partition.

Each L levels that is transmitted is assigned an N-digit binary value. The relationship between N and L is

$$2^N = L$$

Transmitting the binary data requires a distinct pulse shape to be assigned to each bit. Out of the many different ways to achieve this, one method is to assign a negative pulse to any bits with the value of 0, and a positive pulse to bits with the value of 1. If the signal went through a level 16 quantization, each sample of this signal would now be represented by a group of four binary pulses, known as pulse codes, and result in a binary signal.

The message signal $m(t)$ should be band-limited to B Hz, the minimum bandwidth required for the transmission should be $2B$ samples per second. Which means that a bandwidth of $2NB$ bits/s is required.

Some pulses may be detected incorrectly at the receiver due to two sources of errors. Namely, the pulse detection error and the quantization error. Since pulse detection error is quite small compared to the quantization error, it is ignored; and quantization error is assumed to be the only source of error when analyzing this project. In order to find the signal to noise ratio, SNR, we first need to find the mean square value for the quantization error, or the power of the quantization error, and then the power of the message signal.

$$q(t) = \frac{(\Delta v)^2}{3L^2}$$

$$\overbrace{q^2(t)} = \frac{m_p^2}{3L^2}$$

Where the error lies within the range of $(-\Delta v/2, \Delta v/2)$ since the sample value is approximated by the midpoint of the subinterval. SNR can then be found using the following equation.

$$SNR = \frac{S_o}{N_o} = 3L^2 \frac{P_m}{{m_p}^2}$$

where Pm is the power of the message signal $m(t)$, and mp is the peak amplitude. SNR represents the quality of quality of the received signal. We can multiply the SNR by $10log_{10}$ in order to obtain the dB unit. Ideally, it should have a constant SNR for all values of the message signal power. However, the SNR is directly proportional to the signal power, which varies from devices to devices due to the variating values of signal power. This is the result of having uniform value $\Delta v = 2mp/L$ for the quantizing steps. The problem can be resolved by introducing non-uniform quantizing, which uses smaller steps for smaller amplitudes.

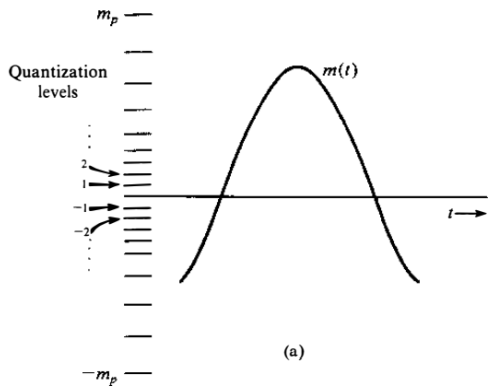

Figure 3a. Nonuniform Quantization

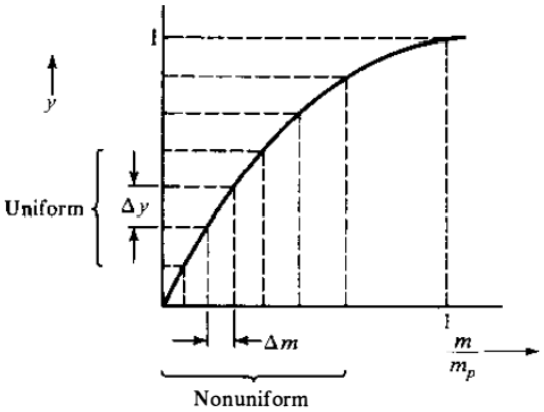

Figure 3b. Input-Output Characteristics of Compressor

Referring to figure 3b, its horizontal axis is the normalized input signal and its vertical axis being the output

signal y. The compressor maps the input signal increments Δm into larger increment Δy for small input signals, or vice versa, in order to yield a quantization noise that is nearly proportional to the signal power. Thus, making SNR more independent of the input signal power.

B. DIFFERENTIAL PULSE CODE MODULATION

PCM is often not very efficient due to the many bits it generates, as well as requiring so much bandwidth to transmit. Since in analog messages, a good estimate or guess can be made about a sample value from the knowledge of past sample values. Meaning that the sample values are not independent, and that there is generally a great deal of redundancy in the Nyquist samples. Exploiting this redundancy, new methods were created to encode the signal with fewer bits; thus, introducing a new scheme called differential pulse code modulation or DPCM. Instead of transmitting the sample values, only the difference d[k] between the successive sample values are transmitted.

If m[k] is the kth sample, then

$$d[k] = m[k] - m[k-1]$$

is transmitted rather than m[k] itself. Having the value of d[k] and the few previous sample values m[k-1], the original signal m[k] can be reconstructed. Since d[k] is a much smaller value than the sample values, the peak amplitude of the transmitted message is reduced significantly. According to $\Delta v = 2mp/L$, the quantization interval is also reduced, thus reducing the quantization noise. Meaning that for a given transmission bandwidth, we are able to increase the SNR; or for a given SNR, we are able to reduce the transmission bandwidth.

To improve upon that scheme, we estimate or predict the value of the kth sample m[k] by using the knowledge of the previous sample values. Let $\hat{m}[k]$ be the estimated sample at k, we are then able to transmit the prediction error $d_p[k]$ using the following equation.

$$d_p[k] = m[k] - \hat{m}[k]$$

The predicted value $\hat{m}[k]$ should be close to $m[k]$ if the prediction was done correctly, and the prediction error $d_p[k]$ should also be smaller than the original $d[k]$. While the prediction error is transmitted, the receiver end will generate $\hat{m}[k]$ from the past sample values and reconstruct the original signal m[k] by adding the predicted error $d_p[k]$ to the estimated signal $\hat{m}[k]$.

$$m[k] = d_p[k] + \hat{m}[k]$$

Rather than receiving than receiving the predicted error $d_p[k]$, and the past sample values m[k-1], m[k-2], etc. We have their quantized version $m_q[k-2], m_q[k-1]$, ... instead. Which means that a quantized version of the estimated signal $\hat{m}_q[k]$ is determined instead of $\hat{m}[k]$. In this case, the predictor error that should be transmitted via PCM is now

$$d_p[k] = m[k] - \hat{m}_q[k]$$

The predictor error is quantized to yield

$$d_q[k] = d[k] + q[k]$$

where q[k] is the quantization error. The predictor output $\hat{m}_q[k]$ is fed back to its input so that the predictor input $m_q[k]$ is

$$\begin{aligned} m_q[k] &= \hat{m}_q[k] + d_q[k] \\ &= m[k] - d[k] + d_q[k] \\ &= m[k] + q[k] \end{aligned}$$

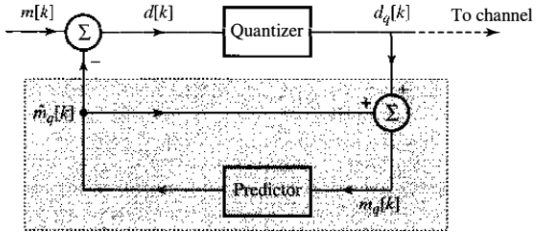

Figure 4a. DPCM Transmitting Block Diagram

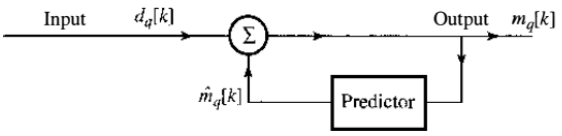

Figure 4b. DPCM Receiving Block Diagram

Then at the receiver end, which has a feedback that is identical to the transmitter due to the inputs in both cases are the same, $d_q[k]$. Meaning that the predictor output must be $\hat{m}_q[k]$ for both the transmitter and the receiver, showing that we receive the desired signal plus the quantization noise associated with the predictor error. The received signal $m_q[k]$ can then be decoded and passed through a low-pass filter for digital to analog conversion.

SNR for DPCM is calculated as the average power of signal over the average power of noise. Where the noise is the average power of DPCM output subtracted from the average power of the original signal, only taking the magnitudes of the noise into consideration.

To determine the improvement of DPCM over PCM, the SNR improvement due to prediction can be found using

$$G_p = \frac{P_m}{P_d}$$

Where P_d is the power of m(t) and P_d is the power of d(t). In terms of decibel units, this means the SNR increases by $10\log_{10}(G_p)$.

C. Line Encoding

Line coding is the process of converting the digital output of a source encoder into the electrical pulses in order to transmit the data over a channel. Line coding is also called transmission coding. This method of transmission has various versions that yields different characteristics like power efficiencies or error detections. Variations of line coding includes on-off, polar, bipolar, and more. Depending on the transmission rate, R_b, the time interval between each pulse is differed since the interval between each pulse is $T_b = 1/R_b$. Moreover, line codes should have the properties of the following:

- *Transmission bandwidth should be as small as possible. Power efficiency should be as low as possible for a given bandwidth and a specific detection error rate.*
- *Errors are desirable to detect, and preferably corrected.*
- *Power spectral density at f = 0 is desired to be zero due to the AC coupling and transformer often used at the repeaters.*
- *Adequate timing content so it is possible to extract timing or clock information from the signal.*
- *Transparency where it becomes possible to correctly transmit a digital signal regardless of the patterns of bits (1s and 0s). A code is transparent if the coded signal for every possible sequence of data is received faithfully.*

In this we report, we will focus on analyzing on-off, and bipolar line code due to their application in this project. On-off line code is the simplest variation, where a 1 in a binary value is transmitted by a pulse, p(t); while a 0 in binary value is transmitted by no pulse. This variation is also known as unipolar line code.

Bipolar line code, also known as pseudo-ternary or alternate mark inversion, is where 0 is encoded by no pulse, and a 1 is encoded by either a positive pulse, p(t), or a negative pulse, -p(t). Where the polarity of the current pulse is determined by the polarity of the previous pulse. Essentially, each pulse should be alternating in polarity. This method of line code provides an advantage where if a single error is made in the detection of pulses, the remaining pulse sequence will violate the bipolar rule. Thus, detecting an error immediately even though it cannot be corrected. Depending on the width of the pulses, there are two schemes of signals. If it's a half width pulse (A full width is a single unit in the x axis in the examples in this report), which has a change to return to the origin, or zero, before the next pulse is called return-to-zero, or RZ. It it's a full width pulse, which does not have a chance to go to zero before the next pulse, and holds a constant value throughout the pulse interval, is called non-return-to-zero, or NRZ.

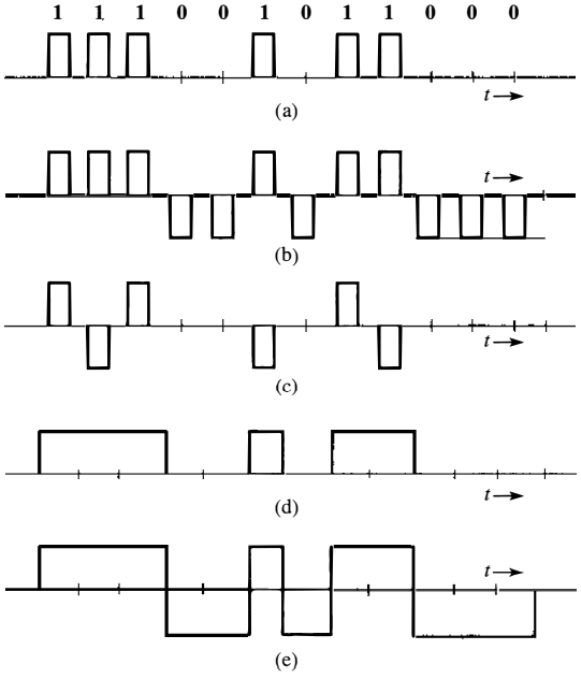

Figure 5. Different line code schemes. (a) on-off (RZ) (b) polar (RZ) (c) bipolar (RZ) (d) on-off (NRZ) (e) polar (NRZ)

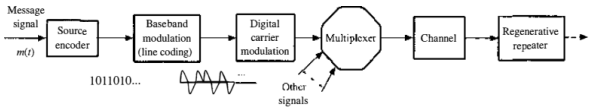

Figure 6. Block diagram of a digital communication system.

In order to achieve zero intersymbol interference (ISI), a raised cosine filter is applied. An ISI isn't necessarily noise, but the spreading of a pulse beyond its allotted timeinterval. A pulse at time t causing interferences with its neighboring pulses at t+n, or t-n. A raised cosine filter is used for pulse shaping where it allows the signal to have a non zero amplitude at its center, and zero amplitudes elsewhere. This allows sucessive transmission of pulses with minimal, controlled ISI or distortions.

D. Modulation Schemes

Signal spectrums are often shifted to a high frequency range when the signal is being transmitted due to the sizable power requirement needed when transmitting at low

frequencies. Spectrum shifted to a higher frequency also transmit various messages simultaneously by sharing the large bandwidth of the transmission medium. Modulation is applied to the spectrum of a signal in order to shift it to a higher frequency by applying the baseband digital signal to modulate a high frequency sinusoid carrier. In order to transmit and receive those shifted information, modulator and demodulators are needed. There are various modulation schemes to transmit digital data, ASK, PSK, and FSK.

ASK, or amplitude shift keying, transmit digital signal by multiplying the pulse wave with a fixed amplitude carrier wave for the duration of the pulse. The on-off baseband m(t) and the carrier modulated signal can be written as

$$m(t) = \sum a_k p(t - kT_b),$$

$$\varphi_{\text{ASK}}(t) = m(t) \cos \omega_c t$$

Where a_k is either a 1, or 0 (On-Off) as shown in figure 7.

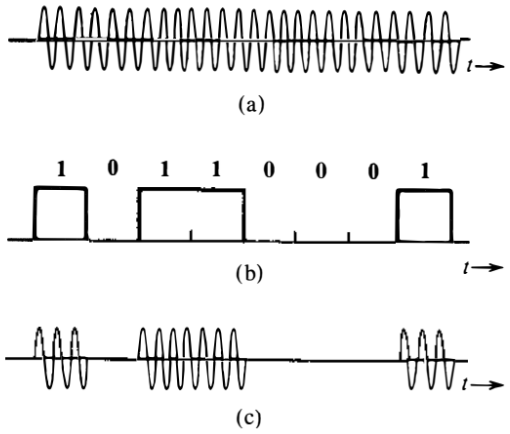

Figure 7a. Demonstrates the ASK process with an on-off line code. Also known as On-Off Keying. (a) Carrier frequency (b) Oon off linecode (c) Modulated ASK signal

ASK can be demodulated both coherently using synchronous detection, or noncoherenly by using the envelope detection.

PSK, also known as phase shift keying, transmits a 1 by with $p(t)\cos(\omega_c t)$ and a 0 by $-p(t)\cos(\omega_c t) = p(t)\cos(\omega_c t + \pi)$. Thus, separating the two pulses by π radian in phase where the information is stored in the phase or the sign of the pulses. PSK can be detected by using coherent detector. The modulated carrier signal can be shown in the same form as ASK.

$$\varphi_{\text{PSK}}(t) = m(t) \cos \omega_c t \qquad m(t) = \sum a_k p(t - kT_b)$$

FSK, or Frequency Shift Keying, is the process of storing the transmitted information in the carrier frequency where a 0 in binary s transmitted by a pulse with the frequency ω_{c0}, and 1 in binary is transmitted by a pulse with the frequency ω_{c1}. FSK is the sum of two interleaved ASK signals with different modulating frequencies (ω_{c0} and ω_{c1}). Its signal is demodulated the same way as ASK, either by synchronous detection or by envelope detection, as it's viewed as two interleaved ASK signals. The received signal is applied to a pair of filters tuned to ω_{c0} and ω_{c1} and are followed by a pair of envelope detectors. Comparing the outputs from both detectors at time t, if a 0 is transmitted, then the pulse will appear at the output of the detector corresponding to $\omega_{c0,}$ while simultaneously, there is no visible signal output on the detector for ω_{c1}. Meaning that a 0 is transmitted at time t. The opposite happens in the case of transmitting a 1.

$$\varphi_{\text{FSK}}(t) = \sum a_k p(t - kT_b) \cos \omega_{c_1} t + \sum (1 - a_k) p(t - kT_b) \cos \omega_{c_0} t$$

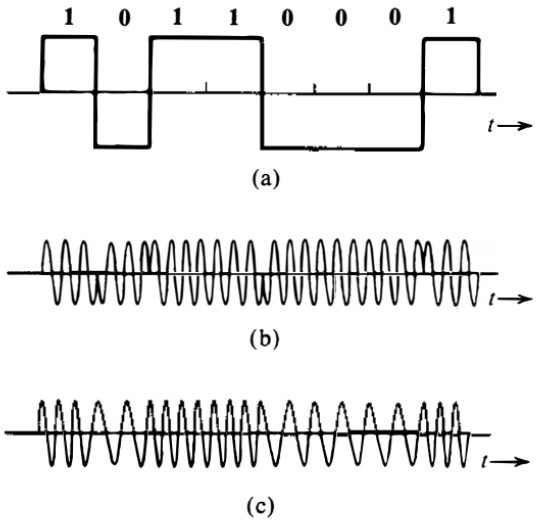

Figure 7b. (a) Polar line code (b) PSK: the modulated signal (c) FSK: the modulated signal

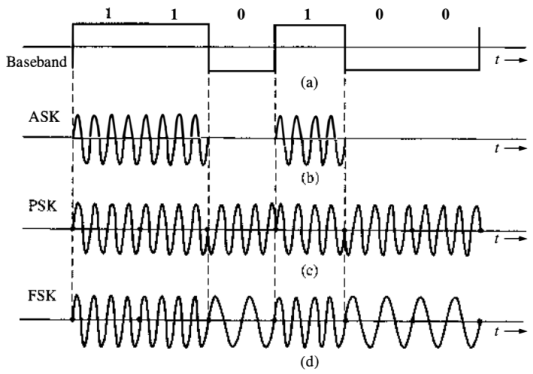

Figure 7c. Digital Modulated Waveforms

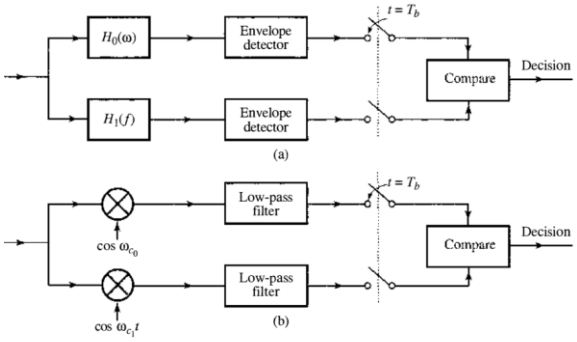

Figure 8. Demodulation: (a)Noncoherent detection of FSK (b) Coherent detection of FSK

III. Procedures

There are five parts to this project, each part building on the previous part. All code mentioned are included in the appendix of this document.

A. The image

The first part of this project focuses on the property of the Lena image provided. The students are to answer the following questions regarding the image.

- What is the size of the image?
- How many bits are being used to represent each pixel?
- What is the range of pixel value?
- Plot the histogram of the input image. Comment on anything interesting.

Utilizing Matlab's functionality, the function imfinfo('lena512colo.tiff') to display the information regarding the Lena image. The size of the image can be found under 'FileSize' or simply read the dimension of the image. The number of bits used to present each pixel is represented as 'BitDepths' and the range of the pixel value can be deducted from its minimum and maximum sample value. To plot the histogram, we first need to read the image using the function imread('filename'), and then plot the histogram using the function histogram(x). The image was also broken down into its YUV format, where we will use the Y components only for the rest of the project past part A.

```
RGB = imread('lena512color.tiff');  %Read the video
info = imfinfo('lena512color.tiff');


%Y component of the Image
R = RGB(:,:,1);
G = RGB(:,:,2);
B = RGB(:,:,3);
Y = 0.299 * R + 0.587 * G + 0.114 * B;
U = -0.14713 * R - 0.28886 * G + 0.436 * B;
V = 0.615 * R - 0.51499 * G - 0.10001 * B;
YUV = cat(3,Y,U,V);        %Concatenate Arrays
3x512x512
YUV1= Y;

%PART A HISTOGRAM
figure;
histogram(Y);
title('Y Component Histogram');
figure;
histogram(RGB);
title('RBG Component Histogram');
```

B. PCM

Second section of this project is to apply PCM with a quantization level of 16, changing the raw Lena image to 4bits per pixel. Tasks for this section includes:

- Plot the histogram of the quantized signal.
- Calculate the SNR at the output of the quantizer.
- Show the quantized image.
- Explain the results from task 2 and 3.

Please note that from here on now, only the Y component (greyscale) of the image is used, instead of all 3 dimension of the image. From now on, the 'image' refers t the Y component of the image only.

Recall that the first step to PCM is to sample the signal, which was done in part A, where the image was read and processed into its different components. In order to manipulate the image, the image matrix was remapped into a row vector. Changing its dimension from [m x n] = [512 x 512] to [1 x (m*n)] = [1x262144]. This was done to organize all pixel values in one row using the code:

```
%ALL pixel values in one row
```

```
X(:,:) = reshape(p(:,:),[1,(m*n)]);
```

Where p, the image, is remapped into the new row vector, X. Now, the second step of PCM, quantization was applied to the remapped signal X. Two methods of quantization were applied, one was developed after researching and exploring with various methods. The other was taken from the book under the Matlab examples. This was done to validate the result of both methods.

The methods developed:

Recall to the theory section regarding quantization, the amplitude of each sample should be the midpoint value of their corresponding partition. The image we use have a range of pixel values going from 0 to 255. This method of quantization splits the range of [0 255] into 16 sections, going from [0, 15], [17, 32], [33, 47], …, [225, 239], [241, 255]. Where all pixel values falling into the range of [0, 15] will be turned into the value of midpoint of this subinterval, 8. Pixel values within [17, 32] will be set to 24, so on and so forth. This is partially demonstrated in the code below. Note that ers1 is the input signal being quantized, and ers1q is the output signal after quantization.

```
for colCount = 1:length(err)
    if (err(rowCount,colCount)>= 0 &&
err(rowCount,colCount)<= 15)
        err1q(rowCount,colCount) = 8;
    elseif (err(rowCount,colCount)>= 17 &&
err(rowCount,colCount)<= 32)
        err1q(rowCount,colCount) = 24;
    elseif (err(rowCount,colCount)>= 33 &&
err(rowCount,colCount)<= 47)
        err1q(rowCount,colCount) = 40;
    elseif err(rowCount,colCount)>= 49 &&
err(rowCount,colCount)<= 64
        err1q(rowCount,colCount) = 56;
end
end
```

The following method taken from the book:

The codes taken from the book was a Matlab example titled “PCM Illustration” on page. 364. It demonstrates the uniform quantization of an analog signal.

```
sig_in = X;                %Assign input sign
L = 16;                    %Assign L level
sig_pmax =max(sig_in) ;    %Finds positive peak of input
signal
sig_nmax =min( sig_in ) ;  %finding the negative peak of
input signal
Delta = ( sig_pmax- sig_nmax )/L; %quantization interval
q_level =sig_nmax+Delta/2 : Delta: sig_pmax-Delta/2 ;
%defines Q-levels
L_sig=length( sig_in ) ;   %find signal length
sigp= ( sig_in- sig_nmax ) / Delta+1/2 ; % convert period
into 1/2 to L+l/2 range
qindex=round( sigp ) ; % round to 1, 2, ... L levels
qindex=min( qindex , L ) ; % eleminate L+l as a rare
possibility
q_out =q_level( qindex ) ; % use index vector to generate
output
eOut = sig_in-q_out;
eOut = round(eOut);
SQNR=20 * log10( norm(sig_in )/norm (eOut)); %actual
SQNR value
```

SNR for both results were calculated using the formula:

$$SNR = 20\log_{10}[\text{ mag}(X)/\text{ mag}(X\text{-Y})]$$

Where mag(X) is the magnitude of the input signal X, and mag(X-Y) is the magnitude of error, or difference, between the input X and the output Y. Note that this method of finding the SNR is a different approach from the book as mentioned earlier in the PCM Theory section. The book shows multiplying the ratio by $10\log_{10}$ to obtain the decibel unit since it calculates the power the SNR using the power of the signal.

C. DPCM

Third section of this project required students to apply DPCM to the image using a 1st order linear predictor and a 3rd order linear predictor, with 4-bit quantizer for the different signals.

Tasks:

- What is the transmitted signal? Plot the histogram of the transmitted signal with correct label.
- Calculate the SNR of the DPCM system.
- Calculate the SNR improvement over the PCM.
- Reconstruct the image on the receiver side and show it.
- Replace the 1st order predictor with a 3rd order predictor (refer to class notes). Repeat task 1 – 4.
- Compare and then explain the results of the two predictors

1st Order Linear Predictor:

Since DPCM requires predicting the samples at t based on previous sample values, the estimated signal $\hat{x}$ must be

derived. First, we have to find the predictor coefficients, which is denoted as a and b in

$$\hat{X} = aY + b$$

Where $\hat{X}$ is the predicted values of the current pixel. Ideally, $\hat{X}$ should be equal to Y, which means a and b should be the smallest values possible:

$$\text{Min E } [(X - \hat{X})^2] = \text{Min E } [(X - aY + b)^2]$$

In order to minimize the predictor coefficient a, we take the first derivative of E $[(X - aY + b)^2]$ with respect to a and set it equal to 0,

$$\frac{d}{da} E[(X - aY - b)^2] = 0$$

Resulting in

$$0 = 2E[(X - aY + b)(-Y)]$$

$$= -\text{E[XY]} + a\text{E[Y2]} + b\text{E[Y]}$$

$$= a\text{E[Y2]} + b\text{E[Y]} = \text{E[XY]} \qquad (5)$$

For b,

$$\frac{d}{db} E[(X - aY - b)^2] = 0$$

$$0 = 2\text{E } [\text{ (X} - a\text{Y} - b) (-1)]$$

$$= -\text{ E[X]} + a\text{ E[Y]} + b$$

$$= a\text{ E[Y]} + b = \text{E[X]}$$

$$b = E[x] - aE[Y] \qquad (6)$$

Substituting equation (6) into (5), we obtain the following:

$$a\text{ E[Y}^2\text{]} + [\text{ E[X]} - a\text{ E[Y]]}\text{ E[Y]} = \text{E[XY]}$$

$$a[\text{E[Y}^2\text{]} + \text{E}^2\text{[Y]]} = \text{E[XY]} - \text{E[X] E[Y]}$$

$$\text{E[XY]} - \text{E[X] E[Y]} = \text{Cov(X, Y)}$$

$$\text{a } = (\text{E[XY]} - \text{E[X] E[Y]}) / ([\text{E[Y}^2\text{]} + \text{E}^2\text{[Y]]})$$

$$= \text{Cov(X, Y)} / (\sigma\text{Y})^2$$

Now substituting a into equation (6), we have

$$b = E[x] - (\text{Cov(X, Y)} / (\sigma\text{Y})^2)E[Y]$$

Since

$$\text{cov(x,y)} = \rho\sigma x\sigma y$$

we now have

$$a = \rho\sigma_x\sigma_y / \sigma_y^2$$

$$b = \text{E[X]} - \rho\sigma_x\sigma_y\text{E[Y]}/\sigma_y^2$$

Thus, obtaining

$$\hat{X} = \rho\sigma_x\sigma_y Y/\sigma_y^2 + \text{E[X]} - \rho\sigma_x\sigma_y\text{E[Y]}/\sigma_y^2$$

$$= \rho\sigma_x(Y - \text{E[Y]})/\sigma_y + \text{E[X]}$$

Matlab Implementation:

Before we implement the derivation above, we must create a new row vector to store the shifted version of the image. This serves as Y from the derivation, this will provide the result of X = Y since they are the same image with only a pixel shifted to the right.

```
%Store the shifted values of X in Y
for count = 1:((m*n)-1)
   Ylinear(1,count) = X(1,count+1);
end
Ylinear(1,262144) = X(1,1);    %Set the last value of Y2 = X(1)
```

Now implementing the derivation of the estimated signal:

```
XY = double(X.*Ylinear);    %Multiplying the X and Y arrays together
%Expected Values
E_x(1,1) = mean(X(:,:));
E_y(1,1) = mean(Ylinear(:,:));
E_xy(1,1) = mean(XY(:,:));

%calculating the prediction coefficients
sigX(1,1) = std(X(:,:));
sigY(1,1) = std(Ylinear(:,:));
rho(1,1) = (E_xy(1,1) - (E_x(1,1) * E_y(1,1)))/((sigX(1,1) * sigY(1,1)));

%Calculating Xhat
Xhat(1,:) = ((rho(1,1)*(Ylinear(1,:)-E_y(1,1))*sigX(1,1))/sigY(1,1)) + E_x(1,1);
```

Then we find the error signal, which is the difference between the original image and the predicted image/estimated signal.

```
%Error Signal or d[k] = actual - predicted
err = X - Xhat;
```

Now we apply the same quantization methods to the error signal used for PCM from part B. We also used the same SNR

equation used for PCM. To find the SNR improvement over PCM from DPCM, To reconstruct the image, we add the quantized error signal to the estimated signal, $\hat{X}$ and reshape it back into a 512 by 512 matrix using the code below:

```
X_Reconstructed(:,:) = reshape(Xq1(:,:),[m,n]);
```

3rd Order Linear Predictor:

Taking the same approach as the 1st order linear predictor, the estimated signal for this predictor is

$$\hat{X} = aY1 + bY2 + cY3$$

Where Y1 is the pixel previous to the current pixel, Y2 is the pixel above Y1, and Y3 is the pixel directly above the current pixel. Similarly, to the 1st order linear predictor,

$$\text{Min } E\left[(X - \hat{X})^2\right] = \text{Min } E\left[(X - aY1 - bY2 - cY3)^2\right]$$

Using Matlab, we find the determinates of the following

```
D = det([EY12 EY1Y2 EY1Y3; EY1Y2 EY22 EY2Y3;
EY1Y3 EY2Y3 EY32]);

Da = det([EXY1 EY1Y2 EY1Y3; EXY2 EY22 EY2Y3;
EXY3 EY2Y3 EY32]);

Db = det([EY12 EXY1 EY1Y3; EY1Y2 EXY2 EY2Y3;
EY1Y3 EXY3 EY32]);

Dc = det([EY12 EY1Y2 EXY1; EY1Y2 EY22 EXY2;
EY1Y3 EY2Y3 EXY3]);

a = Da/D;

b = Db/D;

c = Dc/D;
```

Upon obtaining those values and calculating $\hat{X}$, we then continue to repeat the steps for quantization and finding the SNR as we did for the 1st order linear predictor.

D. LINE ENCODING

We now use the DPCM result from the 1st order linear predictor, and turn it into on off line code and bipolar line code. Since the quantized error signal can be represented in binary, we must convert the error values from decimal to binary. We first need to round each values in the error signal to turn everything into an integer by using floor(), then we can rectify the signal so that there will be no negative values. Using the Matlab function bin2dec().

```
A = round(eOut); %Rounded all values to an integer
for n = 1: length(A)
   if A(1,n) > 0
      E(1,n) = A(1,n);
   else
      E(1,n) = -1*A(1,n);
   end
end
Ebin = dec2bin(E);
```

Where eOut is the quantized error signal, E is the rectified signal, and Ebin is the binary representation of E and the bit stream.

Since on off line code presents a 1 with a pulse and a 0 with no pulse, no further processing is needed. To plot the bipolar line code, recall that bipolar line code transmits a 1 by a pulse and 0 by no pulse. But the polarity of the pulses representing the 1 in binary should alternates. In order to turn the bit stream into bipolar, we use a simple variable in the code to represent the alternation of the pulses.

```
toggle = 0;
for j=1:length(D)
   if D(j) == 1
      if toggle == 0
         lineOut(j) = 1;
         toggle = 1;
      else
         if toggle == 1
            lineOut(j) = -1;
            toggle = 0;
         end
      end
   else
      if D(j) == 0
         lineOut(j) = 0;
      end
   end
end
```

Where D is the input or the bit stream, lineOut is the output that stores the bipolar line code sequence, and toggle is a variable that changes value from 0 to 1 to notify the system when the next pulse needs to change polarity. If the value of toggle is 0, the pulse is positive; if the value of toggle is 1, the pulse is negative.

ISI, or intersymbol interference, is the event of a pulse spreading into its neighboring pulses. In order to minimize intersymbol interference, a raised cosine filter is applied to the line codes with the help of Matlab's built in Raised Cosine Filtering with the characteristic of:

- Samples per Symbol: 20
- Rolloff Factor: 0.4
- Span of Each Symbol: 1

```
Nsym = 1;          % Filter span in symbol durations
beta = 0.4;        % Roll-off factor
sampsPerSym = 20;  % Upsampling factor

rctFilt = comm.RaisedCosineTransmitFilter(...
  'Shape',                'Normal', ...
  'RolloffFactor',        beta, ...
  'FilterSpanInSymbols',  Nsym, ...
  'OutputSamplesPerSymbol', sampsPerSym);
```

Each peak of the raised cosine filter signal represents a pulse. Raised Cosine Filter allows an amplitude of 1 at the center of the signal, while everywhere else having an amplitude of 0. This allows a transmission without any ISI. Line codes can be converted back to the original signal being transmitted at the receiver side if the receiver knows that the signal being transmitted has been quantized by 4 bits. Simply

E. BFSK

Lastly, we apply BFSK to module the pulses from part D. A modulator and a demodulator was added at the transmitter and receiver side. The Modulated signal was also summed with AWGC (Additive White Gaussian Channel) noise during the transmission. It is specified that the frequency is 1MHz for any 0s and that the frequency for 1s is selected according to the minimum spacing criteria *df = 1 / (2 * pi * Tb)* with the data rate R_b of 1k bits/seconds. The channel is tested for an SNR value of: -20dB, -10dB, 0dB, 10dB, and 20dB. Tasks are of the following:

- Plot the first 20 pulses of the BFSK signal with noise for each SNR case.
- Determine the bit errors for each channel noise case. Plot the number of bit errors vs channel SNR.
- Explain your resultReconstruct the image on the receiver side for each SNR case.
- Show and explain your result.<u>As a bonus</u>, repeat task 1 – 3 with bipolar line code.

Students can opt out of reconstructing the image for this part due to the processing power and time needed for that specific task.

```
Tb = 1/1000; %Rb = 1kHz
f1 = 1*10^6;  %Frequency for bit 0
df = 1*((2*pi*Tb));
f2 = 16; %Frequency for bit 1

% The FSK Signal
FSK = [FSK (bit_stream(ii)==0)*sin(2*pi*f1*t)+...
(bit_stream(ii)==1)*sin(2*pi*f2*t)];

% The Original Digital Signal
bits = [bits (bit_stream(ii)==0)*...
zeros(1,length(t)) + …
(bit_stream(ii)==1)*ones(1,length(t))];
```

This code snippet resides in a for loop where ii is incremented, it takes the bit stream and checks its value. If the value is 0, it will multiply that value by *sin(2*pi*f1*t)* and if the value is a 1, it will be multiplied by *sin(2*pi*f2*t).* Note that f1 and f2 are of different values since 1s and 0s are transmitted using different carrier frequencies. Note that the previous code needs to be modified in order to get the BFSK result of Bipolar line code.

```
f3 =9; %Frequency for pulse = -1

% The FSK Signal
FSK = [FSK (bit_stream(ii)==0)*sin(2*pi*f1*t)+...
(bit_stream(ii)==1)*sin(2*pi*f2*t)+ …
(bit_stream(ii)== -1)*sin(2*pi*9*t)];

% The Original Digital Signal
bits = [bits (bit_stream(ii)==0)*...
zeros(1,length(t)) + …
(bit_stream(ii)==1)*ones(1,length(t))+(bit_stream(ii)==-
1)*ones(1*length(t))*-1];
```

where a new frequency f3 was added to distinguish a pulse p(t) = -1 from the pulse p(t) = 1. The AWGC noise is then added using the following:

```
noiseVariance = 0.0; %Noise variance of AWGN channel
noise = sqrt(noiseVariance)*randn(1,length(FSK));
noisySignal = FSK + noise;

%SNR
X = FSK;
magX = norm(X(:,:));
```

```
magError = norm(X(:,:)- noisySignal(:,:));
SNRlog = 20*log10 (norm(magX)/norm(magError))
```

where noiseVariance is the variable that was adjusted in order to obtain the different SNR values mentioned earlier. Value of SNR is again taken the same way as before. We obtain the noise and added it to the BFSK signal, thus, acquiring the noisy signal. Demodulation of BFSK can be done by using the envelope detection, envelope(x) returns the upper and lower envelope. Or it can be done by multiplying the modulated signal with the same carrier frequencies used during the modulation process. Then take the integration of the resulted signals by calling the function trapz(t,x).

```
s1=y1.*m((t+1):n);
s2=y2.*m((t+1):n);
t4=bp/99:bp/99:bp; % Timing
z1=trapz(t4,s1)    % intregation
z2=trapz(t4,s2) % intregation
```

This allows us to move onto the comparison of the value at time t for both signals. Recall from the theory section of BFSK, if the output of the signal corresponding to the frequency for transmitting a 0 has a pulse while the output of the signal corresponding to the frequency for transmitting a 1 has no pulse, we are able to detect that a 0 was transmitted. Vice versa for the detection of transmitting a 1.

```
%This detects if a bit 1 or bit 0 was
transmitted
if(zz1>A/2)
    a=1;
  else(zz2>A/2)
    a=0;
  end
```

In order to obtain the bit error rates, the values in the BFSK signal and the noisy signal are rectified and rounded to their nearest integers. Then the Matlab function biterr(x,y). This function returns the number of bit errors and the bit error rate (BER) by comparing the array x and y. These values were plotted in a graph illustrating the number of bit errors vs the channel SNR.

```
FSKr = round(FSK);
%____RCFILTER
for n = 1: length(FSKr)
  if FSKr(1,n) > 0
    FSK_BER(1,n) = FSKr(1,n);
  else
    FSK_BER(1,n) = -1*FSKr(1,n);
  end
end
nS_R = round(noisySignal);
for n = 1: length(nS_R)
  if nS_R(1,n) > 0
    nS_BER(1,n) = nS_R(1,n);
  else
    nS_BER(1,n) = -1*nS_R(1,n);
  end
end
%Bit Error
[num, ratio] = biterr(FSK_BER ,nS_BER)
```

Number of bit errors should correspond to the SNR, with it increasing with SNR decreasing since a lower SNR means a signal with higher amount of noise. This will be demonstrated later in results.

IV. RESULTS AND DISCUSSION

A. IMAGE PROPERTIES

Raw Image Properties:

Size of the raw Image: 512x512x3

Bits representing each pixel: 24 (3x8)

Range: 0 to 255

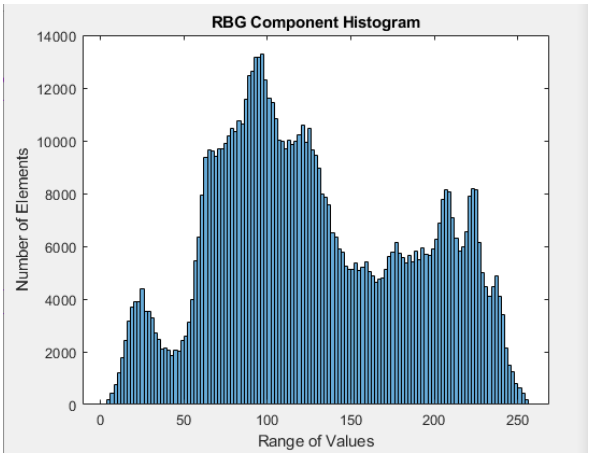

Figure 9a. Histogram of Raw Image

Y Components (Greyscale) Properties:

Size of the Y Component of the Image: 512x512

Bits representing each pixel: 8

Range: 0 to 255

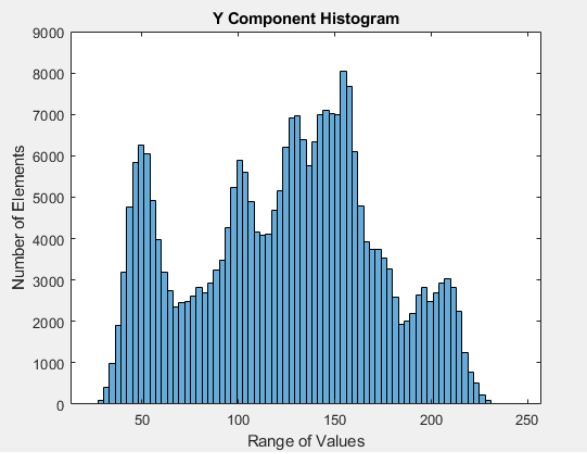

Figure 9b. Histogram of Image in Greyscale

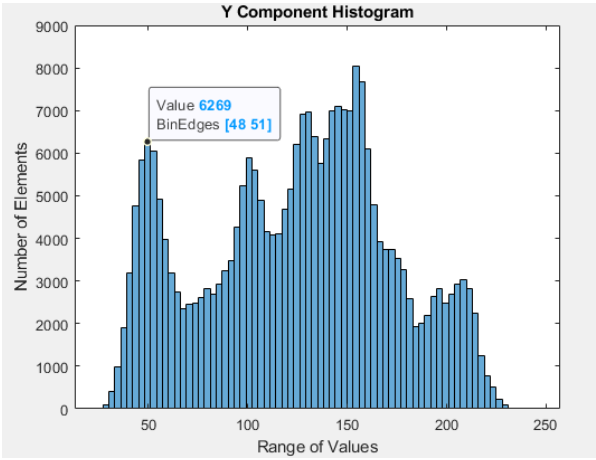

Figure 9c. Histogram of Image in Greyscale with Data

As shown in the procedures of this report, the image was broken down into the YUV format. The 'Y' represents the luma of the image, which is the brightness in an image. While 'U' and 'V' are the chrominance components. 'U' is the blue projection, while 'V' is the red projection. The histogram shows the amount of a certain value exiting in the Y component of the image. In figure Z, there is a total of 6269 pixels that has the value 48, 49, or 50. These histograms have bins with a uniform width. Even though the BinEdges are [48 51], the values of 51 are not included in this bin, but the next bin. In this case, figure Y, most of the values are concentrated towards the center without clipping on the side, meaning that there are no over exposure or under exposure in this image.

Figure 10a. Raw Lena Image

Figure 10b. Y Component of Lena Image.

B. PCM RESULTS

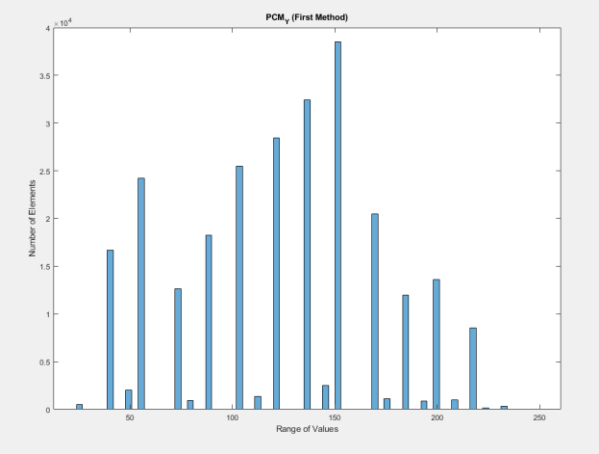

Figure 11a. Histogram of image values after PCM using the first quantization method.

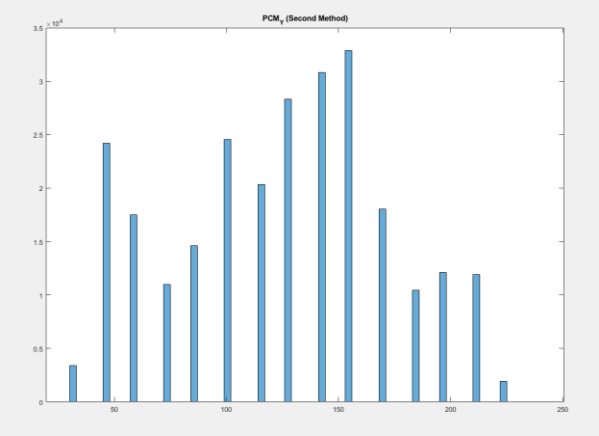

Figure 11b. Histogram of image values after PCM using the second quantization methods.

Table 1. SNR and Error Values from PCM

	SNR	Error
First Method	29.545 dB	2.268e+03
Second Method	30.609 dB	2.006e+03

Figure 12. Reconstructed image after PCM using the second quantization method.

As mentioned earlier, two methods of quantization were applied here. Recall that this quantization should have split the value range of 0 to 255 into 16 sections, where all the values in their corresponding section are set equal to the value of midpoint in such interval. Meaning that there all values should be categorized into 16 different values.

As shown in figure 11a and 11b, the histogram of the first method has some unwanted bins whereas the second method provided a histogram with the ideal number of bins, which is 16. Moreover, using norm() in Matlab, we check the magnitude of the error signals from both methods, the second quantization method provided a lower value in error; as well as having a slightly better SNR value. Shown in table 1.

Since the second method provides a better result, all results will be taken using that quantization method. The reconstructed image figure 12, though not largely, does display some degradation compared to the image before PCM (figure 10b). Checking the actual pixel values, the values are definitely lowered. The SNR of this method was 30.508dB, which is going to be used as a reference in this project as we approach the result of DPCM. There are many standards in the industry, a brief research concludes that an SNR of around 32dB is of excellent image quality. Although in this case, the image looks like it's been noticeably degraded especially when the image transitions from darker pixels to brighter pixels, and should be able to improve upon.

C. DPCM RESULTS

The transmitted signal in DPCM should be the predicted error signal, d_p[k]. Comparing the histograms from both cases, the more predictor coefficient the higher the SNR and lower the error rate of the transmission. The quantized errors are normally distributed among the different values of pixels. As shown in figure 13a and 13b, the values of the quantized error signal using the 1st order linear predictor are mostly distributed among the range of [-50 to 50]. While the values of the quantized error signal using the 3rd order linear predictor is highly concentrated in the range of [0.02, 0.03]. Which shows that most pixel values of the reconstructed image are extremely close to the pixel values of the original image. Overall the values of the error signal rest within the range of [-35, 35].

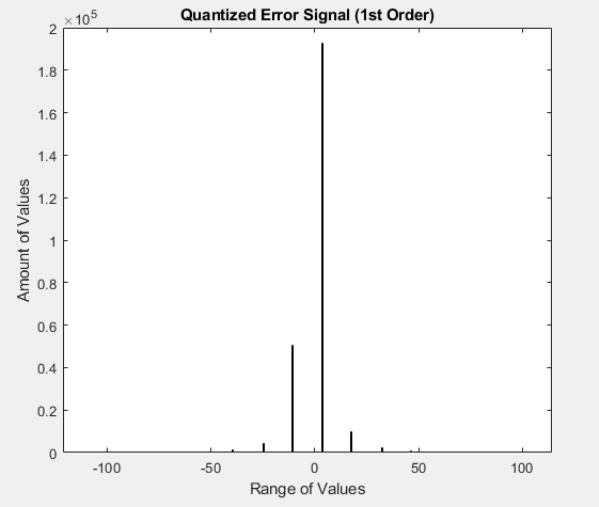

Figure 13a. Transmitted Signal of DPCM (1st Order).

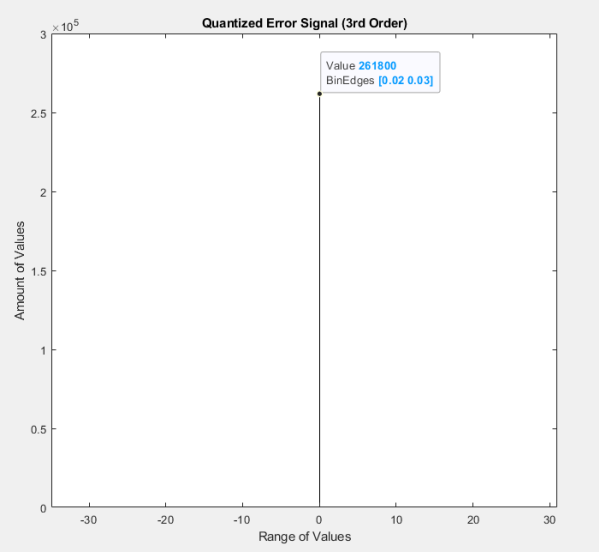

Figure 13b. Transmitted Signal of DPCM (3rd Order).

Figure 14a. Reconstructed Image after DPCM (1st Order).

Figure 14b. Reconstructed Image after DPCM (3rd Order).

Table 2. SNR from DPCM.

	DPCM SNR	SNR Improvement over PCM
1st Order	30.439dB	2.008
3rd Order	67.160dB	3.425

The reconstructed image using the 3rd order linear predictor is slightly closer to the original image than the reconstructed image using the 1st order linear predictor. When comparing reconstructed image, figure 14a and 14b, the difference isn't as obvious as the reconstructed image from PCM. But if we look closely, or check the pixel values of the

images, we can conclude that there is an advantage using the 3rd order linear predictor if we did not have the information from the histograms. Although the SNR of the 1st order DPCM did not improve much over the SNR of PCM, the reconstructed image is significantly better, which creates a sense of false information. To verify the SNR values again, the Matlab function snr(x,y) was used, and yielded extremely similar, if not the same, results. SNR improvement was calculated using the code:

```
%first order SNR
(norm(X).^2)/(norm(q_out).^2)
%third order SNR
(norm(X).^2)/(norm(q_out3).^2)
```

This is a ratio of the magnitude from the original image over the quantized error signal from the DPCM process. DPCM has a high correlation between its current pixel and the pixels near it, the average power of those values are much smaller than the average power of the original image.

As the SNR shows, the 3rd order linear predictor provided a great improvement over the 1st order linear predictor. Upon closer inspection of the histogram of the quantized error signals, error signal from the 1st order linear predictor has about 192500 amount of error values concentrated at lower range of values between [3.5, 4]. 50600 error values in the range of [-11, 10.5], and 9848 error values in the range of [17.5, 18]. Whereas the error signal of the 3rd order linear predictor had 261800 values within the range of [0.02, 0.03]. This leaves 344 other values that are distributed among the values outside of that range of [0.02, 0.03], as shown in figure 15b.

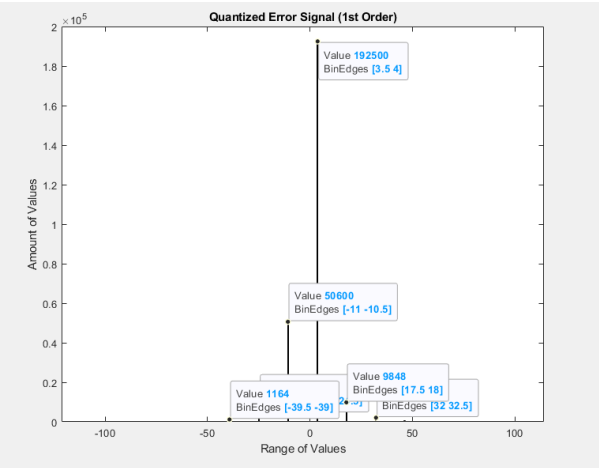

Figure 15a. Histogram with values labeled. (1st Order).

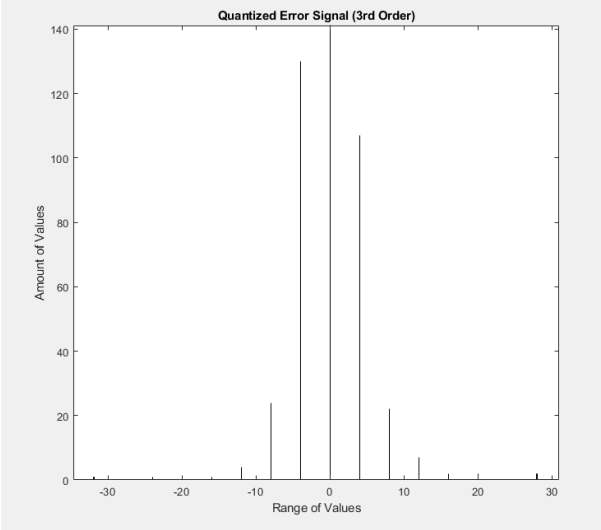

Figure 15b. Histogram with values labeled. (3rd Order).

D. Line Coding Results

Using the parameters that was given to us, we created the raised cosine filter with the following shape:

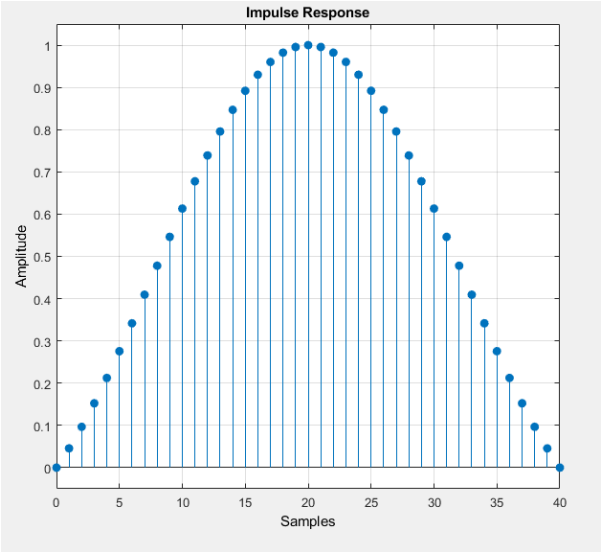

Figure 16. Raised Cosine Filter with Roll-off factor: 0.4, samples per symbol: 20, and span of each samples: 1.

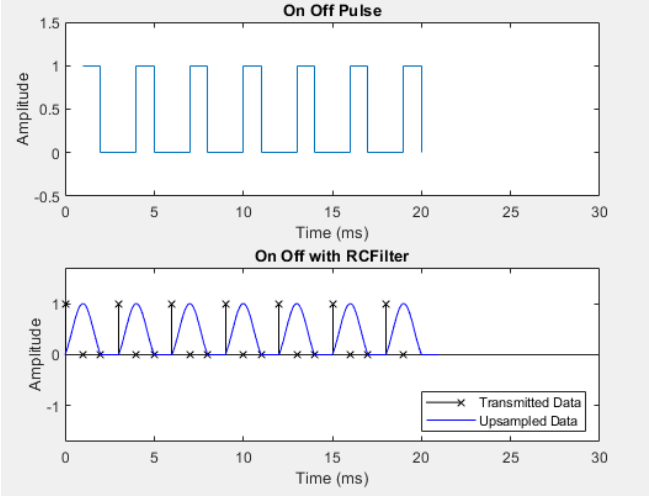

Figure 17a. On-Off Line Code

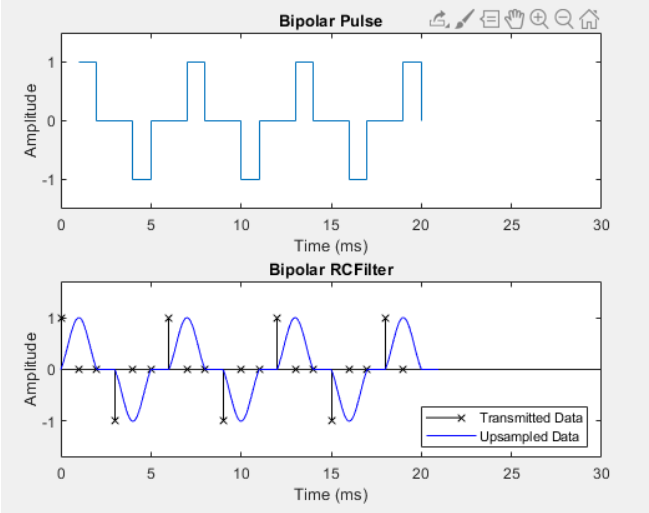

Figure 17b. Bipolar Line Code

Recall that on off line codes transmit the a 1 with a positive pulse and a 0 with no pulse. Bipolar line codes transmitts 1s and 0s similar to on off line code, but the polarity of the pusle alternates every time a 1 is transmitted. Only the first 20 pulses are transmitted. In both figure 17a and 17b, the top subplot is the line code with Raised cosin filter on, and the bottom subplot is the line code itself. Each peaks of the upsampled data from the raised cosine signal represents a pulse.

Upsampling is the process of inserting a value with the magniude of zero between the original samples in order to increase the sampling rate, which increases the resolution rate of the sample and reduces noise. Noise can be created by distortions or ISI, which can be avoided by pulse shaping the input digital signal with the raised cosine filter.

E. BFSK RESULTS

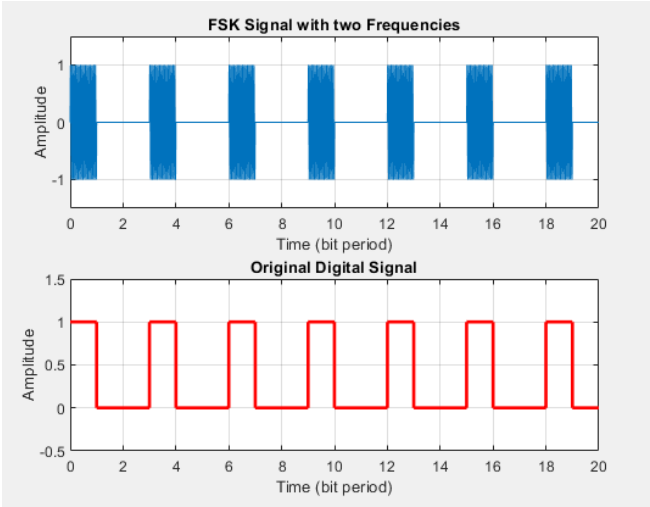

Figure 18a. BFSK signal from on off line code.

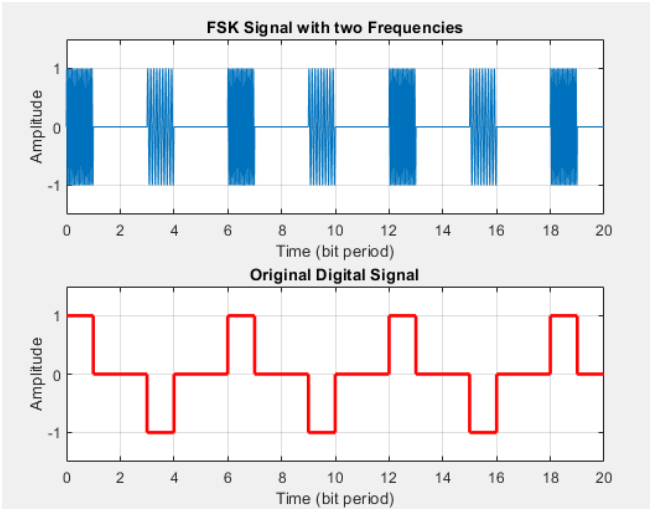

Figure 18b. BFSK signal from

In figure 18a, BFSK of on off code, the frequencies are the same for any pulse with the magnitude of 1. But for BFSK of bipolar line code, there has to be a different frequency to modulate the pulse p(t) = -1. This is demonstrated in figure 1b. This also means that demodulation of this BFSK signal will have a third signal that checks for the pulse p(t) = -1 instead of the two signals that checks for p(t) = 1, 0.

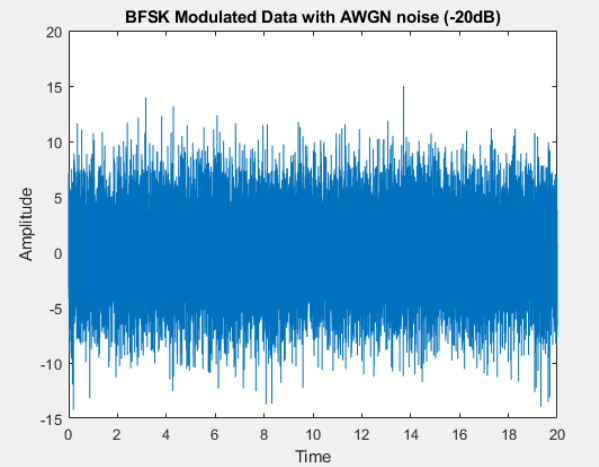

Figure 19a. On Off BFSK with SNR = -20dB.

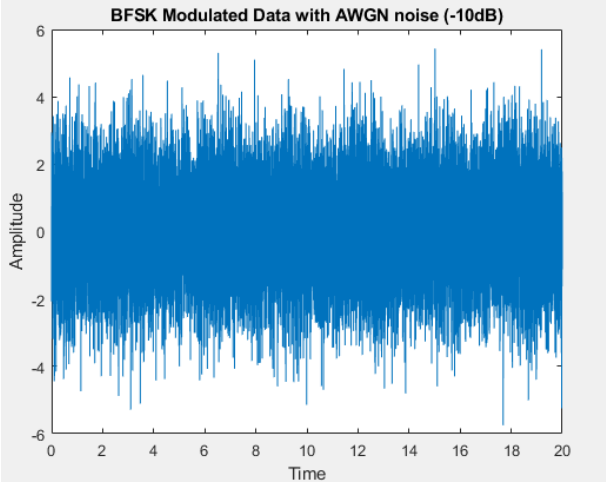

Figure 19b. On Off BFSK with SNR = -10dB.

Figure 19c. On Off BFSK with SNR = 0dB.

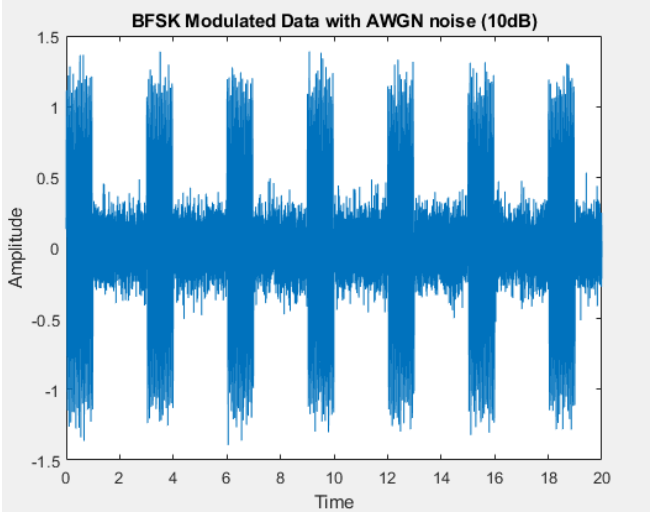

Figure 19d. On Off BFSK with SNR = 10dB.

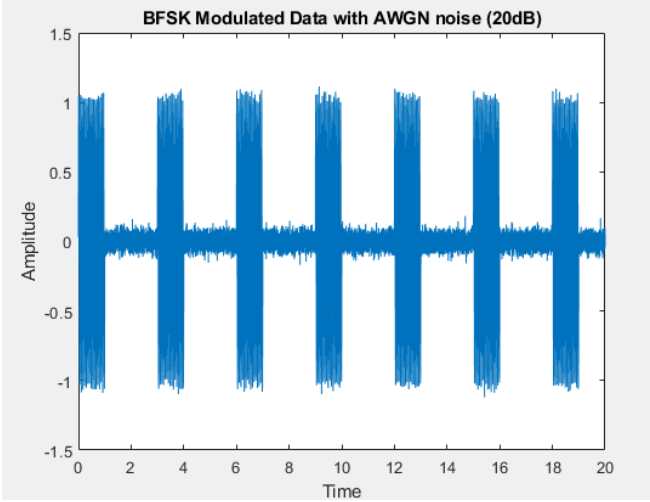

Figure 19e. On Off BFSK with SNR = 20dB.

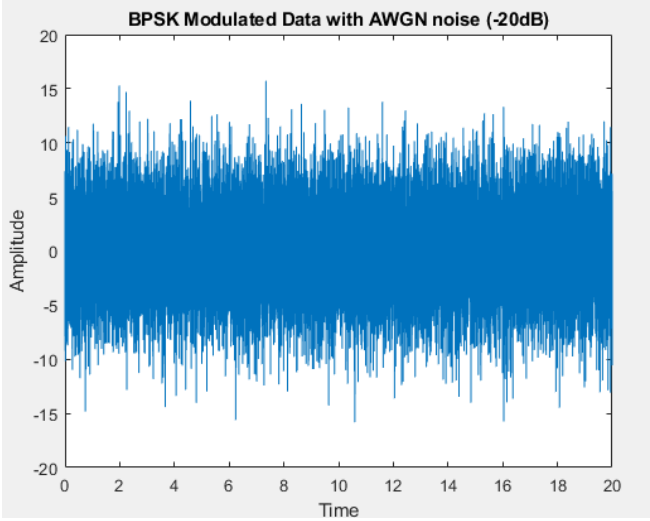

Figure 20a. Bipolar BFSK with SNR = -20dB.

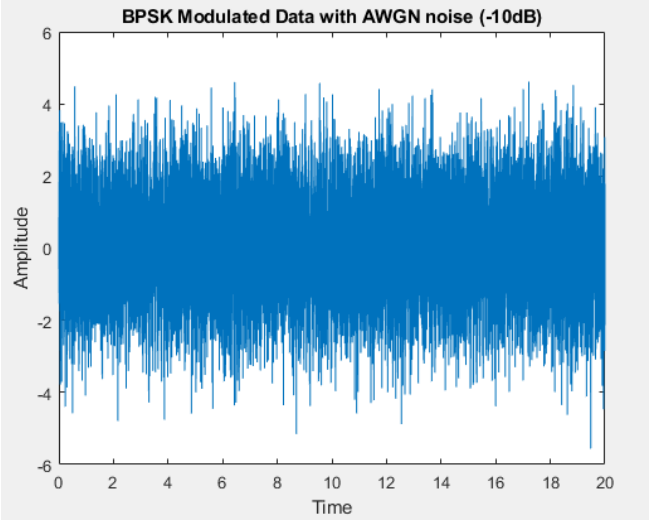

Figure 20b. Bipolar BFSK with SNR = -10dB.

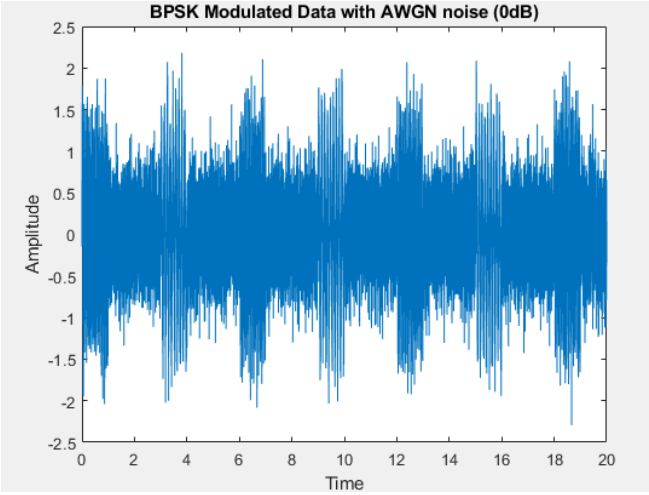

Figure 20c. Bipolar BFSK with SNR = 0dB.

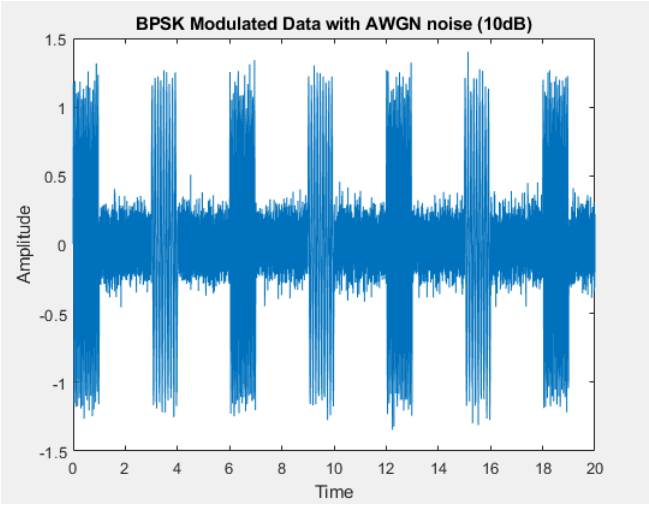

Figure 20d. Bipolar BFSK with SNR = 10dB.

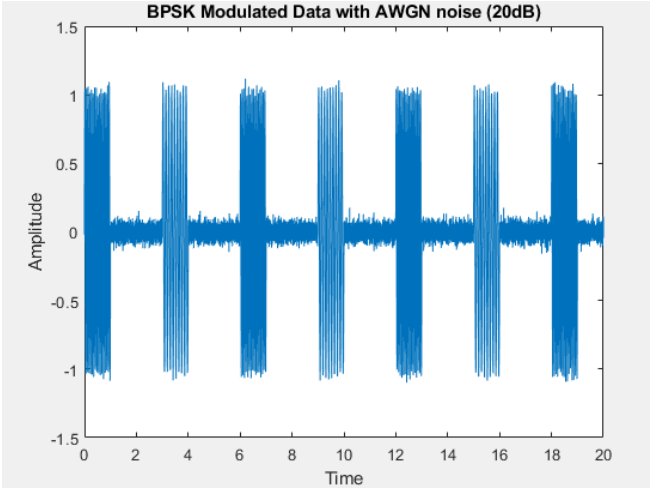

Figure 20e. Bipolar BFSK with SNR = 20dB.

Table 3. SNR and Noise Variance (AWGN)

SNR (dB)	On-Off Noise Variance	Bipolar Noise Variance
-20	18	16
-10	1.8	1.6
0	0.18	0.16
10	0.018	0.016
20	0.0018	0.0016

As expected, the signal becomes noisier with a lower SNR, and clearer with a higher SNR. The 5 cases of SNR tested with the difference of 10dB going from -20dB to 20dB where its noise variance is differed from a factor of 10. Expectedly, lower SNR caused a higher amount of bit errors, demonstrated in table 4, 5 and figure 21a, 21b. The higher the noise variance, more noise there is in the signal. Which isn't surprising, noise variance represents how different the values in the noise signal is. With higher variance, the magnitudes of the noise signal varies more drastically; thus, creating a lower SNR for the whole signal.

The number of bit errors is the number of altered bits in the received data stream over the communication channel due to interference, distortions, or other noises. And the Bit Error Rate, BER, is the presentation of the number of bit errors divided by the total number of bits transferred. For both case, on off and bipolar, the amount of bit errors are quite similar due to them both having the same SNR in each case. The graphs in figure 21a and 21b represents the number of bit errors vs channel SNR. The data of such plots are also stored in table 4 and table 5. A lower SNR means a higher amount of noise in the signal, which corresponds to the higher amount of bit errors in the signals and is demonstrated in the graphs.

Table 4. On Off Bit Error Rate

SNR (dB)	# of bit errors	BER
-20	27374	0.2735
-10	15702	0.2614
0	5533	0.1382
10	581	0.0290
20	168	0.0084

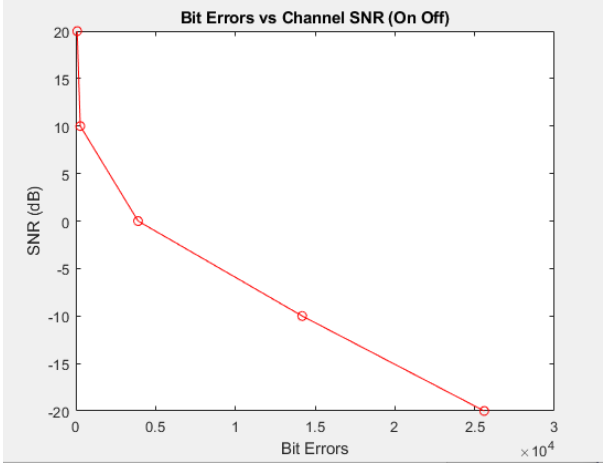

Figure 21a. Bit Errors vs Channel SNR for On Off BFSK

Table 5. Bipolar Bit Error Rate

SNR (dB)	# of bit errors	BER
-20	26789	0.2676
-10	15229	0. 2536
0	5139	0.1283
10	498	0.0249
20	185	0.0092

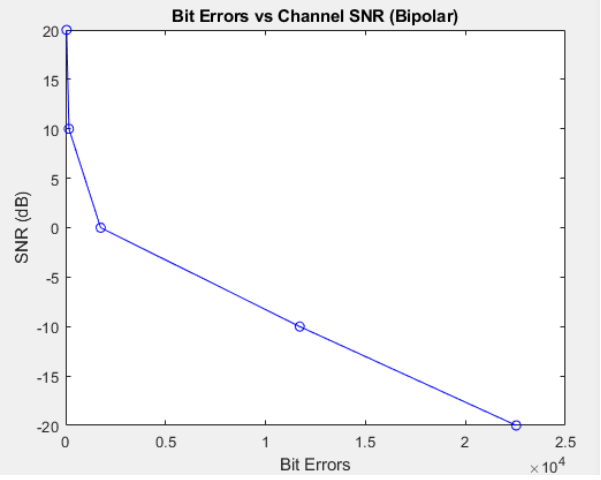

Figure 21b. Bit Errors vs Channel SNR for Bipolar BFSK

V. Conclusion

The purpose of this project was to demonstrate a digital communication system by applying the different industrial schemes on an image. Every step was checked by running a signal to noise ratio calculation to ensure that the difference in quality caused by the various techniques are visible and clear.

Through just part A and part B, it is obvious that DPCM has a significant advantage on PCM. The result from PCM served as baseline for DPCM, to verify where and how the improvements can be made. It can be concluded that the result of DPCM improves with a higher order of linear predictor since the increased amount of predictor coefficients allow for an estimated signal that is better correlated with the original signal. Line encoding demonstrated the different schemes used to convert an analog signal to digital signal, allowing the signal to be manipulated. This ability to manipulate the signal allows other techniques to be applied to in to yield an even better result. So to manipulate the error signal even further, BFSK was implemented and showed that each pulse from the line code is modulated by a carrier frequency in order to achieve maximum efficiency. Further processing the signal, we applied AWGN to the signal to simulate a more real world scenario, which demonstrated the fundamentals of bit errors and BER, and how they are correlated to the SNR of the signal.

VI. References

[1] M. Viswannathan, *SIMULATION OF DIGITAL COMMUNICATION SYSTEMS USING MATLAB*, 2nd ed. Mathuranathan Viswanathan at Amazon.

[2] B. P. Lathi and Z. Ding, *Modern digital and analog communication systems*. Oxford University Press, 2015.

[3] "Raised Cosine Filtering," *Raised Cosine Filtering - MATLAB & Simulink*. [Online]. Available: https://www.mathworks.com/help/comm/examples/raised-cosine-filtering.html. [Accessed: 10-May-2020].

[4] "Binary Frequency Shift Keying," *Binary Frequency Shift Keying - File Exchange - MATLAB Central*. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/30581-binary-frequency-shift-keying. [Accessed: 10-May-2020].

[5] "MATLAB code for FSK modulation and demodulation," MATLAB code for FSK modulation and demodulation - File Exchange - MATLAB Central. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/44821-matlab-code-for-fsk-modulation-and-demodulation. [Accessed: 11-May-2020].

VII. APPENDIX

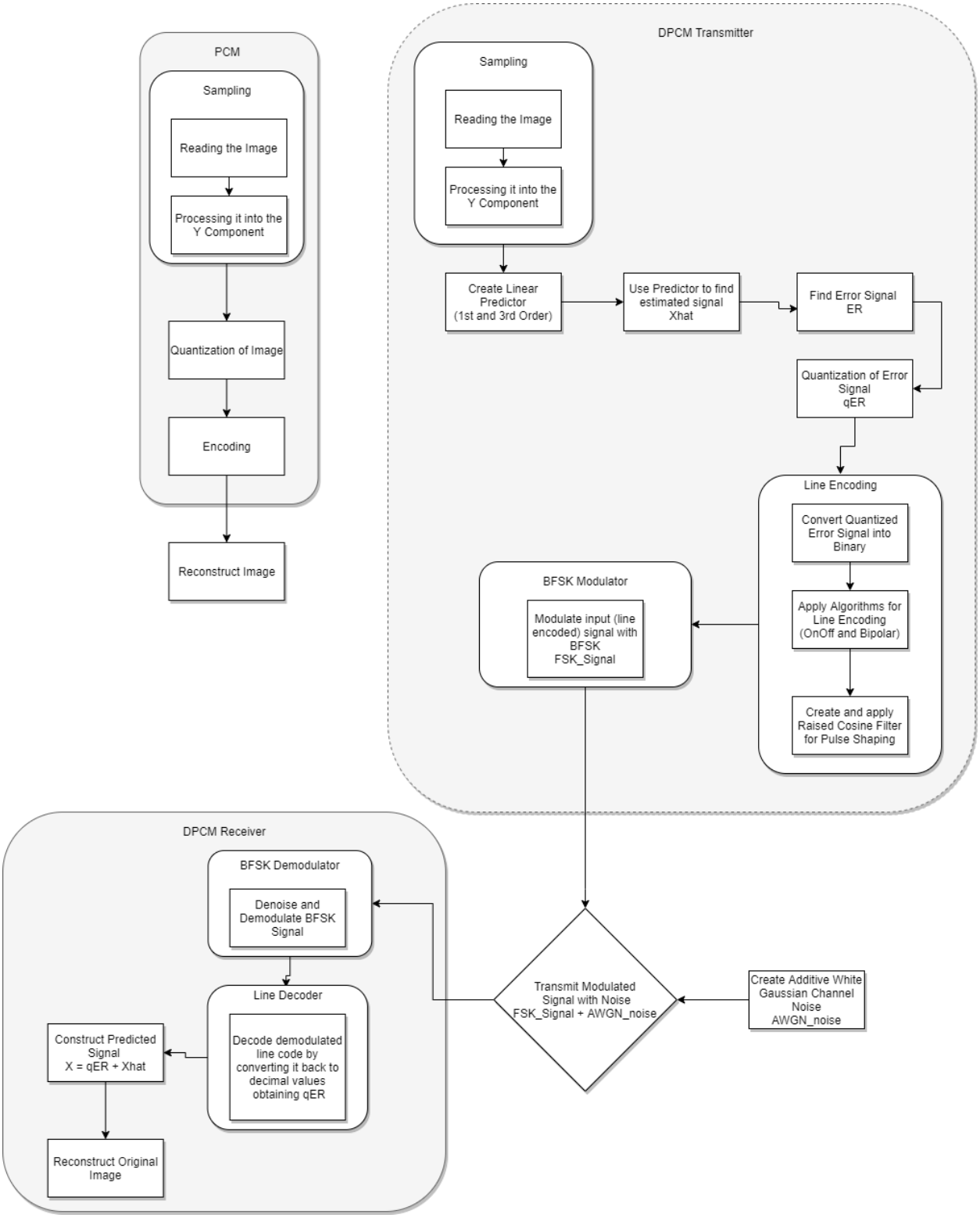

Figure 1. Project Flowchart

PCM Code (Quantization Method Used Throughout the Project):

```
%CHI SHING POON
%1001082535
%EE4330 PROJECT
%TASK: PCM
RGB = imread('lena512color.tiff');  %load the video and converts to double

%Y component of the Image
R = RGB(:,:,1);
G = RGB(:,:,2);
B = RGB(:,:,3);
Y = 0.299 * R + 0.587 * G + 0.114 * B;
U = -0.14713 * R - 0.28886 * G + 0.436 * B;
V = 0.615 * R - 0.51499 * G - 0.10001 * B;
YUV = cat(3,Y,U,V);         %Concatenate Arrays 3x512x512
YUV1= Y;

p = Y;
p = double(p);
[m,n] = size(p(:,:));
%ALL pixel values in one row
X(:,:) = reshape(p(:,:),[1,(m*n)]);

%Quantization
inputSignal = X;                %Assign input sign
L = 16;                     %Assign L level
sig_pmax =max(inputSignal) ;     %Finds positive peak of input signal
sig_nmax =min( inputSignal ) ;   %finding the negative peak of input signal
Delta = ( sig_pmax- sig_nmax )/L; %quantization interval
q_level =sig_nmax+Delta/2 : Delta: sig_pmax-Delta/2 ; %defines Q-levels
L_sig=length( inputSignal ) ;    %find signal length
sigpeak= ( inputSignal- sig_nmax ) / Delta+1/2 ; % convert period into 1/2 to L+l/2 range
qindex=round( sigpeak ) ; % round to 1, 2, ... L levels
qindex=min( qindex , L ) ; % eleminate L+l as a rare possibility
q_out =q_level( qindex ) ; % use index vector to generate output
eOut = inputSignal-q_out;
eOut = round(eOut);
%SNR Calculation
SQNR=20 * log10( norm(inputSignal )/norm (eOut)); %actual SQNR value
%Plot Histogram
figure;
histogram(q_out);
title('PCM_Y');
ylabel('Number of Elements');
xlabel('Range of Values');
%Reshape back to 512 by 512 image matrix
Y_PCM(:,:) = reshape(q_out(:,:),[m,n]);
%Converts variable type back to uint8
PCM_Image = uint8(Y_PCM);
%Display image after quantization
figure;
imshow(PCM_Image);
title('PCM Y');
```

PCM Code(This method of quantization was dropped):

```
%CHI SHING POON
%1001082535
%EE4330 PROJECT
%TASK: Part A and PCM
%Note: Different Quantization method
RGB = imread('lena512color.tiff');  %load the video and converts to double
figure;
imshow(RGB);
title('Raw Image');
%Y component of the Image
R = RGB(:,:,1);
G = RGB(:,:,2);
B = RGB(:,:,3);
Y = 0.299 * R + 0.587 * G + 0.114 * B;
U = -0.14713 * R - 0.28886 * G + 0.436 * B;
V = 0.615 * R - 0.51499 * G - 0.10001 * B;
YUV = cat(3,Y,U,V);         %Concatenate Arrays 3x512x512
YUV1= Y;
%PART A HISTOGRAM
%RGB denser due to having 3 dimen
figure;
histogram(Y);
title('Y Component Histogram');
ylabel('Number of Elements');
xlabel('Range of Values');
figure;
histogram(RGB);
title('RBG Component Histogram');
ylabel('Number of Elements');
xlabel('Range of Values');

p = Y;
p = double(p);
[m,n] = size(p(:,:));
%ALL pixel values in one row
X(:,:) = reshape(p(:,:),[1,(m*n)]);

%PCM Quantization
PCMSELF = X;
rowCount = 1;
for colCount = 1:length(PCMSELF)
    if PCMSELF(rowCount,colCount)>= 0 && PCMSELF(rowCount,colCount)<= 15
        PCMSELF(rowCount,colCount) = 8;
    elseif PCMSELF(rowCount,colCount)>= 17 && PCMSELF(rowCount,colCount)<= 32
        PCMSELF(rowCount,colCount) = 24;
    elseif PCMSELF(rowCount,colCount)>= 33 && PCMSELF(rowCount,colCount)<= 47
        PCMSELF(rowCount,colCount) = 40;
    elseif PCMSELF(rowCount,colCount)>= 49 && PCMSELF(rowCount,colCount)<= 64
        PCMSELF(rowCount,colCount) = 56;
    elseif PCMSELF(rowCount,colCount)>= 65 && PCMSELF(rowCount,colCount)<= 79
        PCMSELF(rowCount,colCount) = 72;
    elseif PCMSELF(rowCount,colCount)>= 81 && PCMSELF(rowCount,colCount)<= 96
        PCMSELF(rowCount,colCount) = 88;
    elseif PCMSELF(rowCount,colCount)>= 97 && PCMSELF(rowCount,colCount)<= 111
        PCMSELF(rowCount,colCount) = 104;
    elseif PCMSELF(rowCount,colCount)>= 113 && PCMSELF(rowCount,colCount)<= 128
        PCMSELF(rowCount,colCount) = 120;
    elseif PCMSELF(rowCount,colCount)>= 129 && PCMSELF(rowCount,colCount)<= 143
        PCMSELF(rowCount,colCount) = 136;
    elseif PCMSELF(rowCount,colCount)>= 145 && PCMSELF(rowCount,colCount)<= 160
```

```
        PCMSELF(rowCount,colCount) = 152;
    elseif PCMSELF(rowCount,colCount)>= 161 && PCMSELF(rowCount,colCount)<= 175
        PCMSELF(rowCount,colCount) = 168;
    elseif PCMSELF(rowCount,colCount)>= 177 && PCMSELF(rowCount,colCount)<= 191
        PCMSELF(rowCount,colCount) = 184;
    elseif PCMSELF(rowCount,colCount)>= 193 && PCMSELF(rowCount,colCount)<= 207
        PCMSELF(rowCount,colCount) = 200;
    elseif PCMSELF(rowCount,colCount)>= 209 && PCMSELF(rowCount,colCount)<= 223
        PCMSELF(rowCount,colCount) = 216;
    elseif PCMSELF(rowCount,colCount)>= 225 && PCMSELF(rowCount,colCount)<= 239
        PCMSELF(rowCount,colCount) = 232;
    elseif PCMSELF(rowCount,colCount)>= 241 && PCMSELF(rowCount,colCount)<= 255
        PCMSELF(rowCount,colCount) = 248;
    elseif PCMSELF(rowCount,colCount)<= 0 && PCMSELF(rowCount,colCount)>= -15
        PCMSELF(rowCount,colCount) = -8;
    elseif PCMSELF(rowCount,colCount)<= -17 && PCMSELF(rowCount,colCount)>= -32
        PCMSELF(rowCount,colCount) = -24;
    elseif PCMSELF(rowCount,colCount)<= -33 && PCMSELF(rowCount,colCount)>= -47
        PCMSELF(rowCount,colCount) = -40;
    elseif PCMSELF(rowCount,colCount)<= -49 && PCMSELF(rowCount,colCount)>= -64
        PCMSELF(rowCount,colCount) = -56;
    elseif PCMSELF(rowCount,colCount)<= -65 && PCMSELF(rowCount,colCount)>= -79
        PCMSELF(rowCount,colCount) = -72;
    elseif PCMSELF(rowCount,colCount)<= -81 && PCMSELF(rowCount,colCount)>= -96
        PCMSELF(rowCount,colCount) = -88;
    elseif PCMSELF(rowCount,colCount)<= -97 && PCMSELF(rowCount,colCount)>= -111
        PCMSELF(rowCount,colCount) = -104;
    elseif PCMSELF(rowCount,colCount)<= -113 && PCMSELF(rowCount,colCount)>= -128
        PCMSELF(rowCount,colCount) = -120;
    elseif PCMSELF(rowCount,colCount)<= -129 && PCMSELF(rowCount,colCount)>= -143
        PCMSELF(rowCount,colCount) = -136;
    elseif PCMSELF(rowCount,colCount)<= -145 && PCMSELF(rowCount,colCount)>= -160
        PCMSELF(rowCount,colCount) = -152;
    elseif PCMSELF(rowCount,colCount)<= -161 && PCMSELF(rowCount,colCount)>= -175
        PCMSELF(rowCount,colCount) = -168;
    elseif PCMSELF(rowCount,colCount)<= -177 && PCMSELF(rowCount,colCount)>= -191
        PCMSELF(rowCount,colCount) = -184;
    elseif PCMSELF(rowCount,colCount)<= -193 && PCMSELF(rowCount,colCount)>= -207
        PCMSELF(rowCount,colCount) = -200;
    elseif PCMSELF(rowCount,colCount)<= -209 && PCMSELF(rowCount,colCount)>= -223
        PCMSELF(rowCount,colCount) = -216;
    elseif PCMSELF(rowCount,colCount)<= -225 && PCMSELF(rowCount,colCount)>= -239
        PCMSELF(rowCount,colCount) = -232;
    elseif PCMSELF(rowCount,colCount)<= -241 && PCMSELF(rowCount,colCount)>= -255
        PCMSELF(rowCount,colCount) = -248;

    end
end

%SNR calculation
SQNR=20*log10( norm(X )/norm (X-PCMSELF));
%Histogram
figure;
histogram(PCMSELF);
title('PCM_Y (First Method)')
ylabel('Number of Elements');
xlabel('Range of Values');

%Reshape back to 512 by 512 image matrix
Y_PCM(:,:) = reshape(PCMSELF(:,:),[m,n]);
%Converts variable type back to uint8
PCM_Image = uint8(Y_PCM);
```

```
%Display image after quantization
figure;
imshow(PCM_Image);
title('PCM Y');
```

DPCM First Order Predictor:

```
%CHI SHING POON
%1001082535
%EE4330 PROJECT
%TASK: DPCM First Order
RGB = imread('lena512color.tiff');  %load the video and converts to double

%Y component of the Image
R = RGB(:,:,1);
G = RGB(:,:,2);
B = RGB(:,:,3);
Y = 0.299 * R + 0.587 * G + 0.114 * B;
U = -0.14713 * R - 0.28886 * G + 0.436 * B;
V = 0.615 * R - 0.51499 * G - 0.10001 * B;
YUV = cat(3,Y,U,V);         %Concatenate Arrays 3x512x512
YUV1= Y;

p = Y;
p = double(p);
[m,n] = size(p(:,:));
%ALL pixel values in one row
X(:,:) = reshape(p(:,:),[1,(m*n)]);

%DPCM 1st
%ORDER-----------------------------------------------------------
%Store the shifted values of X in Y
for count = 1:((m*n)-1)
   Ylinear(1,count) = X(1,count+1);
end
Ylinear(1,262144) = X(1,1);    %Set the last value of Y2 = X(1)

%Xhat Calculation
XY = double(X.*Ylinear);     %Multiplying the X and Y arrays together
%Expected Values
E_x(1,1) = mean(X(:,:));
E_y(1,1) = mean(Ylinear(:,:));
E_xy(1,1) = mean(XY(:,:));

%calculating the prediction coefficients
sigX(1,1) = std(X(:,:));
sigY(1,1) = std(Ylinear(:,:));
rho(1,1) = (E_xy(1,1) - (E_x(1,1) * E_y(1,1)))/((sigX(1,1) * sigY(1,1)));

%Calculating Xhat
Xhat(1,:) = ((rho(1,1)*(Ylinear(1,:)-E_y(1,1))*sigX(1,1))/sigY(1,1)) + E_x(1,1);
bV = (1-rho)*E_y; %Checking Value of b
Xhat = double(Xhat);

%Error Signal or d[k] = actual - predicted
err = X - Xhat;
err2 = err;
err1q = zeros(1,262144);


%%%%%Quantization
sigal_in = err2;
L = 16;
sig_pmax =max(sigal_in) ;
sig_nmax =min( sigal_in ) ; % finding the negative peak
Delta = ( sig_pmax- sig_nmax ) /L; % quantization interval
```

```
q_level =sig_nmax+Delta/2 : Delta: sig_pmax-Delta/2 ; % de f ine Q-levels
L_sig=length( sigal_in ) ; % find signal l ength
sigpeak= ( sigal_in- sig_nmax ) / Delta+1/2 ; % convert into 1/2 to L+ l/2 range
qindex=round( sigpeak ) ; % round to 1, 2, ... L levels
qindex=min( qindex , L ) ; % eleminate L+l as a rare possibility
q_out =q_level( qindex ) ; % use index vec tor to generate output


%plots histogram of quantized error
figure;
% subplot(1,2,1);
% histogram(err1q);
% title('For Loop Quantization DPCM');
% subplot(1,2,2);
histogram(q_out);
title('Quantized Error Signal (1st Order)');
xlabel('Range of Values');
ylabel('Amount of Values');



%Receiver
%From Quantization with better result
XqStep = q_out + Xhat;
X_ReStep(:,:) = reshape(XqStep(:,:),[m,n]);
X_ReStep = uint8(X_ReStep);

%Display Predicted Image
figure;
imshow(X_ReStep);
title('Recontructed after DPCM (1st Order)');
%SNR Calculation
magX = norm(X(:,:));
magErrorS = norm(X(:,:)- XqStep(:,:));
SNR_DCPM1S = 20*log10 (norm(magX)/norm(magErrorS));
```

DPCM Third Order Predictor:

```
%CHI SHING POON
%1001082535
%EE4330 PROJECT
%TASK: DPCM Third Order
RGB = imread('lena512color.tiff');  %load the video and converts to double

%Y component of the Image
R = RGB(:,:,1);
G = RGB(:,:,2);
B = RGB(:,:,3);
Y = 0.299 * R + 0.587 * G + 0.114 * B;
U = -0.14713 * R - 0.28886 * G + 0.436 * B;
V = 0.615 * R - 0.51499 * G - 0.10001 * B;
YUV = cat(3,Y,U,V);         %Concatenate Arrays 3x512x512
YUV1= Y;

p = Y;
p = double(p);
[m,n] = size(p(:,:));
%ALL pixel values in one row
X(:,:) = reshape(p(:,:),[1,(m*n)]);
%Third Order Predictor
%Store the next values of X in Y
%X shifted Once to the right
for count = 1:((m*n)-1)
   y1(1,count) = X(1,count+1);
end
y1(1,262144) = X(1,1);    %Set the last value of Y1 = X(1)

%Y2 is the pixel diagonnally left(n -1, m - 1) of the current pixel
%values are stored in a form of 512x512 then converted to row vector
y1m(:,:) = reshape(y1(:,:),[m,n]);
for rCount = 512:-1: 2
   for mCount = 1: 512
      y2m(rCount,mCount) = y1m(rCount-1, mCount);
   end
end
for mCount = 1:512
   y2m(1,mCount) = y1m(512,mCount);
end
y2 = zeros(size(X));
y2(:,:) = reshape(y2m(:,:),[1,(262144)]);

%y3 is the pixel directly above the current pixel.
%values are stored in a form of 512x512 then converted to row vector
for rCount = 512:-1: 2
   for mCount = 1: 512
      y3m(rCount,mCount) = p(rCount-1, mCount);
   end
end
for mCount = 1:512
   y3m(1,mCount) = p(512,mCount);
end
y3 = zeros(size(X));
y3(:,:) = reshape(y3m(:,:),[1,(262144)]);

% Gets the determinant to find the a,b,c values
D  = (mean2(y1.^2) .*mean2(y2.^2) .*mean2(y3.^2))-(mean2(y1.^2) .*mean2(y2.*y3).*mean2(y2.*y3))...
   -(mean2(y1.*y2).*mean2(y1.*y2).*mean2(y3.^2))+2*(mean2(y1.*y2).*mean2(y2.*y3).*mean2(y1.*y3))...
   -(mean2(y1.*y3).*mean2(y2.^2) .*mean2(y1.*y3));
```

```
Da = (mean2(X.*y1) .*mean2(y2.^2) .*mean2(y3.^2))-(mean2(X.*y1) .*mean2(y2.*y3).*mean2(y2.*y3))-...
    (mean2(y1.*y2).*mean2(X.*y2) .*mean2(y3.^2))+(mean2(y1.*y2).*mean2(y2.*y3).*mean2(X.*y3))+...
    (mean2(y1.*y3).*mean2(X.*y2) .*mean2(y2.*y3))-(mean2(y1.*y3).*mean2(y2.^2) .*mean2(X.*y3));

Db = (mean2(y1.^2) .*mean2(X.*y2) .*mean2(y3.^2))-(mean2(y1.^2) .*mean2(X.*y3) .*mean2(y2.*y3))-...
    (mean2(X.*y1) .*mean2(y1.*y2).*mean2(y3.^2))+(mean2(X.*y1) .*mean2(y1.*y3).*mean2(y2.*y3))+...
    (mean2(y1.*y3).*mean2(y1.*y2).*mean2(X.*y3))-(mean2(y1.*y3).*mean2(X.*y2) .*mean2(y1.*y3));

Dc = (mean2(y1.^2) .*mean2(y2.^2) .*mean2(X.*y3))-(mean2(y1.^2) .*mean2(X.*y2) .*mean2(y2.*y3))-...
    (mean2(y1.*y2).*mean2(y1.*y2).*mean2(X.*y3))+(mean2(y1.*y2).*mean2(X.*y2) .*mean2(y1.*y3))+...
    (mean2(X.*y1) .*mean2(y1.*y2).*mean2(y2.*y3))-(mean2(y2.^2) .*mean2(y1.*y3).*mean2(X.*y1));


%Gets a, b, and c values from cramer's rule
a=Da./D;
b=Db./D;
c=Dc./D;

%Xhat equation
%This X_HAT IS MARGINALLY lower than the Xhat from 1st Order
Xhat3=a.*y1+b.*y2+c.*y3;
e3 = X - Xhat3;
e4= e3;

%Quantization
sigal_in3 = e4;
L = 16;
sig_pmax3 =max(sigal_in3) ;
sig_nmax3 =min( sigal_in3 ) ; % finding the negative peak
Delta3 = ( sig_pmax3- sig_nmax3 ) /L; % quantization interval
q_level3 =sig_nmax3+Delta3/2 : Delta3: sig_pmax3-Delta3/2 ; % de f ine Q-levels
L_sig =length( sigal_in3 ) ; % find signal l ength
sigpeak3 = ( sigal_in3- sig_nmax3 ) / Delta3 +1/2 ; % convert into 1/2 to L+ l/2 range
qindex3=round( sigpeak3 ) ; % round to 1, 2, ... L levels
qindex3=min( qindex3 , L ) ; % eleminate L+l as a rare possibility
q_out3 =q_level3( qindex3 ) ; % use index vec tor to generate output


%Histograms
figure;
histogram(q_out3);
title('Quantized Error Signal (3rd Order)');
xlabel('Range of Values');
ylabel('Amount of Values');



Xq3S = q_out3 + Xhat3;

%SNR
% magError3 = norm(X(:,:)- Xq3(:,:));
% SNR_DCPM3 = 20*log10 (norm(magX)/norm(magError3));
magError3S = norm(X(:,:)- Xq3S(:,:));
SNR_DCPM3S = 20*log10 (norm(magX)/norm(magError3S));

%Step yields a smaller value which makes sense
X_ReStep3(:,:) = reshape(Xq3S(:,:),[m,n]);
X_ReStep3 = uint8(X_ReStep3);
%display Image
figure;
imshow(X_ReStep3);
```

```
title("Reconstructed after DPCM (3rd Order)");


%SNR IMPROVEMENT
(norm(X).^2)/(norm(q_out).^2)
```

On Off Line Code:

```
%CHI SHING POON
%1001082535
%EE4330 PROJECT
%TASK: LineCode ON OFF
RGB = imread('lena512color.tiff');  %load the video and converts to double

%Y component of the Image
R = RGB(:,:,1);
G = RGB(:,:,2);
B = RGB(:,:,3);
Y = 0.299 * R + 0.587 * G + 0.114 * B;
U = -0.14713 * R - 0.28886 * G + 0.436 * B;
V = 0.615 * R - 0.51499 * G - 0.10001 * B;
YUV = cat(3,Y,U,V);         %Concatenate Arrays 3x512x512
YUV1= Y;

p = Y;
p = double(p);
[m,n] = size(p(:,:));
%ALL pixel values in one row
X(:,:) = reshape(p(:,:),[1,(m*n)]);
%DPCM 1st
%ORDER-----------------------------------------------------------
%Store the next values of X in Y
for count = 1:((m*n)-1)
   Ylinear(1,count) = X(1,count+1);
end
Ylinear(1,262144) = X(1,1);    %Set the last value of Y2 = X(1)

%Xhat Calculation
XY = double(X.*Ylinear);     %Multiplying the X and Y arrays together
%Expected Values
E_x(1,1) = mean(X(:,:));
E_y(1,1) = mean(Ylinear(:,:));
E_xy(1,1) = mean(XY(:,:));

%calculating the prediction coefficients
sigX(1,1) = std(X(:,:));
sigY(1,1) = std(Ylinear(:,:));
rho(1,1) = (E_xy(1,1) - (E_x(1,1) * E_y(1,1)))/((sigX(1,1) * sigY(1,1)));

Xhat(1,:) = ((rho(1,1)*(Ylinear(1,:)-E_y(1,1))*sigX(1,1))/sigY(1,1)) + E_x(1,1);
bV = (1-rho)*E_y; %Checking Value of b

Xhat = double(Xhat);
%Error Signal or d[k] = actual - predicted
err1 = X - Xhat;


%%%%%%%%%Quantization%%%%%%%%%
sig_in3 = err1;
L = 16;
sig_pmax3 =max(sig_in3) ;
sig_nmax3 =min( sig_in3 ) ; % finding the negative peak
Delta3 = ( sig_pmax3- sig_nmax3 ) /L; % quantization interval
q_level3 =sig_nmax3+Delta3/2 : Delta3: sig_pmax3-Delta3/2 ; % de f ine Q-levels
L_sig =length( sig_in3 ) ; % find signal l ength
sigp3 = ( sig_in3- sig_nmax3 ) / Delta3 +1/2 ; % convert into 1/2 to L+ l/2 range
qindex3=round( sigp3 ) ; % round to 1, 2, ... L levels
qindex3=min( qindex3 , L ) ; % eleminate L+l as a rare possibility
q_out3 =q_level3( qindex3 ) ; % use index vec tor to generate output
```

```
eOut3 = sig_in3-q_out3;
eOut3 = round(eOut3);


%____RCFILTER
A = round(q_out3); %Rounded all values to an integer
for n = 1: length(A)
   if A(1,n) > 0
      E(1,n) = A(1,n);
   else
      E(1,n) = -1*A(1,n);
   end
end
Ebin = dec2bin(E(1:20));
%Reshape for input
C = reshape(Ebin',1,numel(Ebin)); % converted into bits
D=num2str(C)-'0';


%RCF Parameters
Nsym = 2;          % Filter span in symbol durations
beta = 0.4;        % Roll-off factor
sampsPerSym = 20;    % Upsampling factor
rctFilt = comm.RaisedCosineTransmitFilter(...
  'Shape',                'Normal', ...
  'RolloffFactor',        beta, ...
  'FilterSpanInSymbols',    Nsym, ...
  'OutputSamplesPerSymbol', sampsPerSym);
% Normalize to obtain maximum filter tap value of 1
b = coeffs(rctFilt);
rctFilt.Gain = 1/max(b.Numerator);

% Visualize the impulse response
fvtool(rctFilt, 'Analysis', 'impulse')
% Visualize the impulse response
fvtool(rctFilt, 'Analysis', 'impulse')

% Parameters
DataL = 20;            % Data length in symbols
% DataL = length(Q4E);
R = 1000;              % Data rate
Fs = R * sampsPerSym;   % Sampling frequency

% Create a local random stream to be used by random number generators for
% repeatability
hStr = RandStream('mt19937ar', 'Seed', 0);

% Generate random data
%  x = 2*randi(hStr, [0 1], DataL, 1)-1;
%Bit stream of first twenty pulses
x = zeros(20,1);
for i = 1: 20
   x(i,1) = D(1,i);
end

% Time vector sampled at symbol rate in milliseconds
tx = 1000 * (0: DataL - 1) / R;

% Filter
yo = rctFilt([x; zeros(Nsym/2,1)]);
% Time vector sampled at sampling frequency in milliseconds
to = 1000 * (0: (DataL+Nsym/2)*sampsPerSym - 1) / Fs;
```

```
% Plot data

fig1 = figure;
subplot(2,1,2);
stem(tx, x, 'kx'); hold on;
% Plot filtered data
plot(to, yo, 'b-'); hold off;
% Set axes and labels
axis([0 30 -1.7 1.7]);  xlabel('Time (ms)'); ylabel('Amplitude'); title('On Off with RCFilter');
legend('Transmitted Data', 'Upsampled Data', 'Location', 'southeast')
subplot(2,1,1);
stairs(x)
title('On Off Pulse');
 xlabel('Time (ms)'); ylabel('Amplitude');
ylim([-0.5 1.5]);
xlim([0 30]);
```

Bipolar Line Code:

```
%CHI SHING POON
%1001082535
%EE4330 PROJECT
%TASK: RCF LINE CODE BIPOLAR
RGB = imread('lena512color.tiff');  %load the video and converts to double

%Y component of the Image
R = RGB(:,:,1);
G = RGB(:,:,2);
B = RGB(:,:,3);
Y = 0.299 * R + 0.587 * G + 0.114 * B;
U = -0.14713 * R - 0.28886 * G + 0.436 * B;
V = 0.615 * R - 0.51499 * G - 0.10001 * B;
YUV = cat(3,Y,U,V);         %Concatenate Arrays 3x512x512
YUV1= Y;

p = Y;
p = double(p);
[m,n] = size(p(:,:));
%ALL pixel values in one row
X(:,:) = reshape(p(:,:),[1,(m*n)]);

%DPCM 1st
%ORDER-----------------------------------------------------
%Store the next values of X in Y
for count = 1:((m*n)-1)
   Ylinear(1,count) = X(1,count+1);
end
Ylinear(1,262144) = X(1,1);    %Set the last value of Y2 = X(1)

%Xhat Calculation
XY = double(X.*Ylinear);     %Multiplying the X and Y arrays together
%Expected Values
E_x(1,1) = mean(X(:,:));
E_y(1,1) = mean(Ylinear(:,:));
E_xy(1,1) = mean(XY(:,:));

%calculating the prediction coefficients
sigX(1,1) = std(X(:,:));
sigY(1,1) = std(Ylinear(:,:));
rho(1,1) = (E_xy(1,1) - (E_x(1,1) * E_y(1,1)))/((sigX(1,1) * sigY(1,1)));

Xhat(1,:) = ((rho(1,1)*(Ylinear(1,:)-E_y(1,1))*sigX(1,1))/sigY(1,1)) + E_x(1,1);
bV = (1-rho)*E_y; %Checking Value of b

Xhat = double(Xhat);
%Error Signal or d[k] = actual - predicted
err1 = X - Xhat;


%%%%%%%%%Quantization%%%%%%%%%
sig_in3 = err1;
L = 16;
sig_pmax3 =max(sig_in3) ;
sig_nmax3 =min( sig_in3 ) ; % finding the negative peak
Delta3 = ( sig_pmax3- sig_nmax3 ) /L; % quantization interval
q_level3 =sig_nmax3+Delta3/2 : Delta3: sig_pmax3-Delta3/2 ; % de f ine Q-levels
L_sig =length( sig_in3 ) ; % find signal l ength
sigp3 = ( sig_in3- sig_nmax3 ) / Delta3 +1/2 ; % convert into 1/2 to L+ l/2 range
qindex3=round( sigp3 ) ; % round to 1, 2, ... L levels
qindex3=min( qindex3 , L ) ; % eleminate L+l as a rare possibility
```

```
q_out3 =q_level3( qindex3 ) ; % use index vec tor to generate output
eOut3 = sig_in3-q_out3;
eOut3 = round(eOut3);


%____RCFILTER
A = round(q_out3); %Rounded all values to an integer
for n = 1: length(A)
   if A(1,n) > 0
      E(1,n) = A(1,n);
   else
      E(1,n) = -1*A(1,n);
   end
end
Ebin = dec2bin(E(1:20));

C = reshape(Ebin',1,numel(Ebin)); % converted into bits
%D = str2bin(C);
D=num2str(C)-'0';

F = bin2dec(Ebin);

lineOut = zeros(size(D));
toggle = 0;
for j=1:length(D)
   if D(j) == 1
      if toggle == 0
         lineOut(j) = 1;
         toggle = 1;
      else
         if toggle == 1
            lineOut(j) = -1;
            toggle = 0;
         end
      end
   else
      if D(j) == 0
         lineOut(j) = 0;
      end
   end
end
% figure;
% stairs(lineOut(1:20));

Nsym = 2;          % Filter span in symbol durations
beta = 0.4;        % Roll-off factor
sampsPerSym = 20;    % Upsampling factor
rctFilt = comm.RaisedCosineTransmitFilter(...
  'Shape',                 'Normal', ...
  'RolloffFactor',         beta, ...
  'FilterSpanInSymbols',    Nsym, ...
  'OutputSamplesPerSymbol', sampsPerSym);
% Normalize to obtain maximum filter tap value of 1
b = coeffs(rctFilt);
rctFilt.Gain = 1/max(b.Numerator);

% Visualize the impulse response
fvtool(rctFilt, 'Analysis', 'impulse')
% Visualize the impulse response
fvtool(rctFilt, 'Analysis', 'impulse')

% Parameters
```

```
DataL = 20;            % Data length in symbols
% DataL = length(Q4E);
R = 1000;              % Data rate
Fs = R * sampsPerSym;   % Sampling frequency

% Create a local random stream to be used by random number generators for
% repeatability
hStr = RandStream('mt19937ar', 'Seed', 0);

% Generate random data
%  x = 2*randi(hStr, [0 1], DataL, 1)-1;
x = zeros(20,1);
for i = 1: 20
   x(i,1) = lineOut(1,i);
end
% Time vector sampled at symbol rate in milliseconds
tx = 1000 * (0: DataL - 1) / R;

% Filter
yo = rctFilt([x; zeros(Nsym/2,1)]);
% Time vector sampled at sampling frequency in milliseconds
to = 1000 * (0: (DataL+Nsym/2)*sampsPerSym - 1) / Fs;

% Plot data
fig1 = figure;
subplot(2,1,2);
stem(tx, x, 'kx'); hold on;
% Plot filtered data
plot(to, yo, 'b-'); hold off;
% Set axes and labels
axis([0 30 -1.7 1.7]);  xlabel('Time (ms)'); ylabel('Amplitude');title('Bipolar RCFilter');
legend('Transmitted Data', 'Upsampled Data', 'Location', 'southeast')
subplot(2,1,1);
stairs(x);
title('Bipolar Pulse');
 xlabel('Time (ms)'); ylabel('Amplitude');
ylim([-1.5 1.5]);
xlim([0 30]);

%DECODER%%%%%%%%%%%%%%%%%%%%%%%
% pMax = 1;
% pMin = -1;
% decodeStream =yo;
% tStream = zeros(1, 30);
% tCount = 1;
%
% subplot(3,1,3);
% plot(tStream, decodeStream);
% title('Decoded Signal');
% % xlabel('Time (ms)'); ylabel('Amplitude');
% ylim([-1.5 1.5]);
% xlim([0 30]);
```

BFSK On Off:

```
%CHI SHING POON
%1001082535
%EE4330 PROJECT
%TASK: BFSK ON OFF

RGB = imread('lena512color.tiff');  %load the video and converts to double

%Y component of the Image
```

```
R = RGB(:,:,1);
G = RGB(:,:,2);
B = RGB(:,:,3);
Y = 0.299 * R + 0.587 * G + 0.114 * B;
U = -0.14713 * R - 0.28886 * G + 0.436 * B;
V = 0.615 * R - 0.51499 * G - 0.10001 * B;
YUV = cat(3,Y,U,V);         %Concatenate Arrays 3x512x512
YUV1= Y;

p = Y;
p = double(p);
[m,n] = size(p(:,:));
%ALL pixel values in one row
X(:,:) = reshape(p(:,:),[1,(m*n)]);

%DPCM 1st
%ORDER-----------------------------------------------------
%Store the next values of X in Y
for count = 1:((m*n)-1)
   Ylinear(1,count) = X(1,count+1);
end
Ylinear(1,262144) = X(1,1);    %Set the last value of Y2 = X(1)

%Xhat Calculation
XY = double(X.*Ylinear);     %Multiplying the X and Y arrays together
%Expected Values
E_x(1,1) = mean(X(:,:));
E_y(1,1) = mean(Ylinear(:,:));
E_xy(1,1) = mean(XY(:,:));

%calculating the prediction coefficients
sigX(1,1) = std(X(:,:));
sigY(1,1) = std(Ylinear(:,:));
rho(1,1) = (E_xy(1,1) - (E_x(1,1) * E_y(1,1)))/((sigX(1,1) * sigY(1,1)));

Xhat(1,:) = ((rho(1,1)*(Ylinear(1,:)-E_y(1,1))*sigX(1,1))/sigY(1,1)) + E_x(1,1);
bV = (1-rho)*E_y; %Checking Value of b

Xhat = double(Xhat);
%Error Signal or d[k] = actual - predicted
err1 = X - Xhat;


%%%%%%%%%Quantization%%%%%%%%%
sig_in3 = err1;
L = 16;
sig_pmax3 =max(sig_in3) ;
sig_nmax3 =min( sig_in3 ) ; % finding the negative peak
Delta3 = ( sig_pmax3- sig_nmax3 ) /L; % quantization interval
q_level3 =sig_nmax3+Delta3/2 : Delta3: sig_pmax3-Delta3/2 ; % de f ine Q-levels
L_sig =length( sig_in3 ) ; % find signal l ength
sigp3 = ( sig_in3- sig_nmax3 ) / Delta3 +1/2 ; % convert into 1/2 to L+ l/2 range
qindex3=round( sigp3 ) ; % round to 1, 2, ... L levels
qindex3=min( qindex3 , L ) ; % eleminate L+l as a rare possibility
q_out3 =q_level3( qindex3 ) ; % use index vec tor to generate output
eOut3 = sig_in3-q_out3;
eOut3 = round(eOut3);


%____RCFILTER
A = round(q_out3); %Rounded all values to an integer
for n = 1: length(A)
```

```
    if A(1,n) > 0
        E(1,n) = A(1,n);
    else
        E(1,n) = -1*A(1,n);
    end
end
Ebin = dec2bin(E(1:20));

C = reshape(Ebin',1,numel(Ebin)); % converted into bits
%D = str2bin(C);
D=num2str(C)-'0';
bit_stream = D;
% Enter the two frequencies
% Frequency component for 0 bit
f1 = 1*10^6;
% Frequency component for 1 bit
Rb = 1000;
Tb = 1/Rb; %Rb = 1kHz
df = 1*((2*pi*Tb));
% f2 = f1*df;
f2 = 16;
% Sampling rate - This will define the resoultion
fs = 100;
% Time for one bit
t = 0: Tb : 1;
% This time variable is just for plot
time = [];
FSK = [];
bits = [];
for ii = 1: 1: 20

    % The FSK Signal
    FSK = [FSK (bit_stream(ii)==0)*sin(2*pi*f1*t)+...
        (bit_stream(ii)==1)*sin(2*pi*f2*t)];

    % The Original Digital Signal
    bits = [bits (bit_stream(ii)==0)*...
        zeros(1,length(t)) + (bit_stream(ii)==1)*ones(1,length(t))];

    time = [time t];
    t =  t + 1;

end

% Plot the FSK Signal
subplot(2,1,1);
plot(time,FSK);
xlabel('Time (bit period)');
ylabel('Amplitude');
title('FSK Signal with two Frequencies');
axis([0 time(end) -1.5 1.5]);
grid  on;
% Plot the Original Digital Signal
subplot(2,1,2);
plot(time,bits,'r','LineWidth',2);
xlabel('Time (bit period)');
ylabel('Amplitude');
title('Original Digital Signal');
axis([0 time(end) -0.5 1.5]);
grid on;
```

```
%-------------------------------------------
%Adding Channel Noise
%-------------------------------------------
% noiseOut = awgn(FSK, -20);
noiseVariance = .0018; %Noise variance of AWGN channel

noise = sqrt(noiseVariance)*randn(1,length(FSK));
noisySignal = FSK + noise;
% figure;
% plot(noiseOut);
% title('Matla AWGN');
figure;
plot(time,noisySignal);
xlabel('Time');
ylabel('Amplitude');
title('BFSK Modulated Data with AWGN noise (20dB)');
% ylim([-1.5 1.5]);
Rb = 1000;
Fs=16*Rb;
% Fs = fs;
Tb =1/Rb;

%SNR 20log
X1 = FSK;
magX = norm(X1(:,:));
magError = norm(X1(:,:)- noisySignal(:,:));
SNR = 20*log10 (norm(magX)/norm(magError));

%Used for BER
FSKr = round(FSK);
%____RCFILTER
for n = 1: length(FSKr)
   if FSKr(1,n) > 0
      FSK_BER(1,n) = FSKr(1,n);
   else
      FSK_BER(1,n) = -1*FSKr(1,n);
   end
end
nS_R = round(noisySignal);
for n = 1: length(nS_R)
   if nS_R(1,n) > 0
      nS_BER(1,n) = nS_R(1,n);
   else
      nS_BER(1,n) = -1*nS_R(1,n);
   end
end


%Bit Error Calculation
[num, ratio] = biterr(FSK_BER ,nS_BER)
```

BFSK Bipolar:

```
%CHI SHING POON
%1001082535
%EE4330 PROJECT
%TASK: BFSK BIPOLAR
%CHANGE NOISE VARIANCE TO ADJUST SNR
RGB = imread('lena512color.tiff');  %load the video and converts to double

%Y component of the Image
```

```
R = RGB(:,:,1);
G = RGB(:,:,2);
B = RGB(:,:,3);
Y = 0.299 * R + 0.587 * G + 0.114 * B;
U = -0.14713 * R - 0.28886 * G + 0.436 * B;
V = 0.615 * R - 0.51499 * G - 0.10001 * B;
YUV = cat(3,Y,U,V);         %Concatenate Arrays 3x512x512
YUV1= Y;

p = Y;
p = double(p);
[m,n] = size(p(:,:));
%ALL pixel values in one row
X(:,:) = reshape(p(:,:),[1,(m*n)]);

%DPCM 1st
%ORDER-----------------------------------------------------
%Store the next values of X in Y
for count = 1:((m*n)-1)
   Ylinear(1,count) = X(1,count+1);
end
Ylinear(1,262144) = X(1,1);    %Set the last value of Y2 = X(1)

%Xhat Calculation
XY = double(X.*Ylinear);     %Multiplying the X and Y arrays together
%Expected Values
E_x(1,1) = mean(X(:,:));
E_y(1,1) = mean(Ylinear(:,:));
E_xy(1,1) = mean(XY(:,:));

%calculating the prediction coefficients
sigX(1,1) = std(X(:,:));
sigY(1,1) = std(Ylinear(:,:));
rho(1,1) = (E_xy(1,1) - (E_x(1,1) * E_y(1,1)))/((sigX(1,1) * sigY(1,1)));

Xhat(1,:) = ((rho(1,1)*(Ylinear(1,:)-E_y(1,1))*sigX(1,1))/sigY(1,1)) + E_x(1,1);
bV = (1-rho)*E_y; %Checking Value of b

Xhat = double(Xhat);
%Error Signal or d[k] = actual - predicted
err1 = X - Xhat;


%%%%%%%%%Quantization%%%%%%%%%
sig_in3 = err1;
L = 16;
sig_pmax3 =max(sig_in3) ;
sig_nmax3 =min( sig_in3 ) ; % finding the negative peak
Delta3 = ( sig_pmax3- sig_nmax3 ) /L; % quantization interval
q_level3 =sig_nmax3+Delta3/2 : Delta3: sig_pmax3-Delta3/2 ; % de f ine Q-levels
L_sig =length( sig_in3 ) ; % find signal l ength
sigp3 = ( sig_in3- sig_nmax3 ) / Delta3 +1/2 ; % convert into 1/2 to L+ l/2 range
qindex3=round( sigp3 ) ; % round to 1, 2, ... L levels
qindex3=min( qindex3 , L ) ; % eleminate L+l as a rare possibility
q_out3 =q_level3( qindex3 ) ; % use index vec tor to generate output
eOut3 = sig_in3-q_out3;
eOut3 = round(eOut3);


%____RCFILTER
A = round(q_out3); %Rounded all values to an integer
for n = 1: length(A)
```

```
    if A(1,n) > 0
        E(1,n) = A(1,n);
    else
        E(1,n) = -1*A(1,n);
    end
end
Ebin = dec2bin(E(1:20));

C = reshape(Ebin',1,numel(Ebin)); % converted into bits
%D = str2bin(C);
D=num2str(C)-'0';

F = bin2dec(Ebin);

lineOut = zeros(size(D));
toggle = 0;
for j=1:length(D)
    if D(j) == 1
        if toggle == 0
            lineOut(j) = 1;
            toggle = 1;
        else
            if toggle == 1
                lineOut(j) = -1;
                toggle = 0;
            end
        end
    else
        if D(j) == 0
            lineOut(j) = 0;
        end
    end
end
bit_stream = lineOut;
figure;
% Enter the two frequencies
% Frequency component for 0 bit
f1 = 1*10^6;
% Frequency component for 1 bit
Rb = 1000;
Tb = 1/Rb; %Rb = 1kHz
df = 1*((2*pi*Tb));
% f2 = f1*df;
f2 = 16;
f3 = 9;
% Sampling rate - This will define the resoultion
fs = 100;

% Time for one bit
t = 0: Tb : 1;
% This time variable is just for plot
time = [];
FSK = [];
bits = [];
for ii = 1: 1: 20

    % The FSK Signal
    FSK = [FSK (bit_stream(ii)==0)*sin(2*pi*f1*t)+...
        (bit_stream(ii)==1)*sin(2*pi*f2*t)+(bit_stream(ii)==-1)*sin(2*pi*9*t)];

    % The Original Digital Signal
    bits = [bits (bit_stream(ii)==0)*...
```

```
        zeros(1,length(t)) + (bit_stream(ii)==1)*ones(1,length(t))+(bit_stream(ii)==-1)*ones(1*length(t))*-1];

    time = [time t];
    t =  t + 1;

end

% Plot the FSK Signal
subplot(2,1,1);
plot(time,FSK);
xlabel('Time (bit period)');
ylabel('Amplitude');
title('FSK Signal with two Frequencies');
axis([0 time(end) -1.5 1.5]);
grid  on;
% Plot the Original Digital Signal
subplot(2,1,2);
plot(time,bits,'r','LineWidth',2);
xlabel('Time (bit period)');
ylabel('Amplitude');
title('Original Digital Signal');
axis([0 time(end) -1.5 1.5]);
grid on;



%-----------------------------------------
%Adding Channel Noise
%-----------------------------------------

noiseVariance = .0016; %Noise variance of AWGN channel
noise = sqrt(noiseVariance)*randn(1,length(FSK));
noisySignal = FSK + noise;
figure;
plot(time,noisySignal);
xlabel('Time');
ylabel('Amplitude');
title('BPSK Modulated Data with AWGN noise (-20dB)');
% ylim([-1.5 1.5]);
Rb = 1000;
Fs=16*Rb;
% Fs = fs;
Tb =1/Rb;

%SNR 20log
X1 = FSK;
magX = norm(X1(:,:));
magError = norm(X1(:,:)- noisySignal(:,:));
SNR = 20*log10 (norm(magX)/norm(magError));

%BER Calculation
%____RCFILTER
FSKr = round(FSK);
for n = 1: length(FSKr)
    if FSKr(1,n) > 0
        FSK_BER(1,n) = FSKr(1,n);
    else
        FSK_BER(1,n) = -1*FSKr(1,n);
    end
end
nS_R = round(noisySignal);
for n = 1: length(nS_R)
```

```
    if nS_R(1,n) > 0
        nS_BER(1,n) = nS_R(1,n);
    else
        nS_BER(1,n) = -1*nS_R(1,n);
    end
end
%Bit Error
[num, ratio] = biterr(FSK_BER ,nS_BER)

% %Demodulation Using th Envelope Method
% % deFSK = [];
% % for ii = 1: 1: length(FSK)
% %
% %     % The FSK Signal
% %     deFSK = [deFSK (FSK(ii)==0)*sin(2*pi*f1*t)+...
% %         (FSK(ii)==1)*sin(2*pi*f2*t+df)];
% %
% % %     time = [time t];
% % %     t =  t + 1;
% %
% % end
% [up, low] = envelope(FSK());
% %
% % figure;
% % title('Envelope Detection');
% % xlabel('Time');
% % ylabel('Amplitude');
% % hold on;
% % plot(time, up,time, low);
% % hold off;
% %
% figure;
% title('Envelope Detection with half wave rectifier');
% xlabel('Time');
% ylabel('Amplitude');
% hold on;
% plot(time, up,time,low);
% hold off;
% % dem = FSK.*sin(2*pi*f2*time);
% % z1=trapz(time, dem);
% % zz1=round(2*z1/1);
% % figure;
% % plot(zz1);
% % dem = [];
% % for i = 1 :length(up)
% %     if up(i) >= 1       %Threshold for a positive pulse
% %         dem(i) = 1;
% %     else dem(i) = 0;
% %     end
% end
% %
% % figure;
% % stairs(dem);
```